

WEBMASTERS



Microsoft
Silverlight™
Développez des applications web
riches et plus interactives !
2

 **Micro**
Application

Copyright © 2009 Micro Application - 20-22, rue des Petits-Hôtels - 75010 Paris

1^{ère} Édition - Février 2009

Auteurs - Loïc BAR, Simon BOIGELOT

Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (article L122-4 du code de la propriété intellectuelle).

Avertissement aux utilisateurs

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles L335-2 et suivants du code de la propriété intellectuelle.

Le code de la propriété intellectuelle n'autorise, aux termes de l'article L122-5, que les reproductions strictement destinées à l'usage privé et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et courtes citations dans un but d'exemple et d'illustration.

Les informations contenues dans cet ouvrage sont données à titre indicatif et n'ont aucun caractère exhaustif voire certain. A titre d'exemple non limitatif, cet ouvrage peut vous proposer une ou plusieurs adresses de sites Web qui ne seront plus d'actualité ou dont le contenu aura changé au moment où vous en prendrez connaissance.

Aussi, ces informations ne sauraient engager la responsabilité de l'Editeur. La société MICRO APPLICATION ne pourra être tenue pour responsable de toute omission, erreur ou lacune qui aurait pu se glisser dans cet ouvrage ainsi que des conséquences, quelles qu'elles soient, qui résulteraient des informations et indications fournies ainsi que de leur utilisation.

Tous les produits cités dans cet ouvrage sont protégés, et les marques déposées par leurs titulaires de droits respectifs. Cet ouvrage n'est ni édité, ni produit par le(s) propriétaire(s) de(s) programme(s) sur le(s)quel(s) il porte et les marques ne sont utilisées qu'à seule fin de désignation des produits en tant que noms de ces derniers.

ISBN : 978-2-300-014314

Couverture réalisée par Sébastien Wiegant

MICRO APPLICATION
20-22, rue des Petits-Hôtels
75010 PARIS
Tél. : 01 53 34 20 20
Fax : 01 53 34 20 00

Support technique :
Également disponible sur www.microapp.com
<http://www.microapp.com>

Retrouvez des informations sur cet ouvrage !

Rendez-vous sur le site **Internet de Micro Application** www.microapp.com. Dans le module de recherche, sur la page d'accueil du site, entrez la référence à 4 chiffres indiquée sur le présent livre. Vous accédez directement à sa fiche produit.



→ RECHERCHE

1431 OK

Tous les produits ▼

Avant-propos

La collection *Webmasters* s'adresse aux personnes initiées au développement de sites web qui souhaitent découvrir et mettre en pratique les nouvelles technologies Internet. Sans négliger les aspects théoriques, nous donnons toujours priorité à la pratique afin que vous puissiez rapidement être autonome.

À travers les différents titres de cette collection vous découvrirez les technologies qui font le web 2.0 et feront ce que certains nomment déjà le web 3.0.

Conventions typographiques

Afin de faciliter la compréhension des techniques décrites, nous avons adopté les conventions typographiques suivantes :

- **gras** : menu, commande, boîte de dialogue, bouton, onglet.
- *italique* : zone de texte, liste déroulante, case à cocher, bouton radio.
- Police bâton : instruction, listing, texte à saisir.
- ➡ : dans les scripts, indique un retour à la ligne volontaire dû aux contraintes de la mise en page.



Il s'agit d'informations complémentaires relatives au sujet traité. Propose conseils et trucs pratiques.



Mise en garde sur un point important à ne pas négliger.

Sommaire

1 << Le langage XAML	22
1.1 Introduction	24
1.2 Les bases de XAML	24
Héritage de XML	24
Adaptation de XML	25
1.3 Les éléments de structure	26
Grid	26
StackPanel	30
Canvas	31
ScrollViewer	32
Border	33
1.4 Les éléments de contenu	35
Images	35
TextBlock	36
ProgressBar	37
1.5 Les événements et leur traitement	38
1.6 Les éléments d'interactions	40
Button	40
CheckBox	42
ToggleButton	43
RadioButton	45
TextBox	46
PasswordBox	47
ListBox et ListBoxItem	47
ComboBox et ComboBoxItem	49
Slider	51
1.7 Autres éléments utiles	51
Line	51
Rectangle	52
Popup	53
1.8 Première approche du DataBinding	55
DataTemplates	61
ValueConverter	66
Le fichier Generic.Xaml	72
Redéfinir la structure d'une ListBox	72
1.9 Colorez votre application grâce aux Brushes et aux Gradients ..	73

1.10	Animez votre application grâce aux StoryBoard	79
	Créez une bannière Silverlight grâce aux animations	82
1.11	Check-List	95

2 << Créer vos applications avec Expression Studio 96

2.1	Introduction à Expression Studio	98
2.2	Expression Design	99
2.3	Expression Encoder 2	102
2.4	Expression Blend 2	104
2.5	Intéraction entre Expression Blend et Visual Studio 2008	109
2.6	Check-List	111

3 << Exploiter vos sources de données 112

3.1	Utilisez SQL et votre base de données	114
	Silverlight, C# et SQL Serveur : introduction	114
	SQL	115
	Les commandes SQL en C#	116
3.2	Exploitez vos données sur Oracle	120
3.3	MySQL et Silverlight	124
3.4	LINQ	126
	LINQ, un peu d'explication	129
	LINQ to XML par l'exemple	129
3.5	Les Web services	134
3.6	ADO.NET/Silverlight	143
3.7	Créez un widget météo	153
	MapCodesToConditions	170
3.8	Traitez un flux de données RSS	174
3.9	Check-list	179

4 << Silverlight et ASP.NET 180

4.1	Introduction à ASP.NET	182
	ASP.NET	182
	Prérequis	183

	Premier exemple	183
	Le Web.config	190
4.2	Les contrôles ASP.NET	191
	Les contrôles standard	191
	Les contrôles de validation	192
	Les contrôles riches	193
	Les contrôles de données	193
	Les contrôles de navigation	195
	Les contrôles de login	196
	Les contrôles HTML	197
	Postback et ViewState	197
4.3	Les contrôles ASP.NET pour Silverlight	198
	Le contrôle MediaPlayer	201
	MediaPlayer et JavaScript	205
	Le contrôle Silverlight	207
4.4	Interaction de Silverlight avec la page	210
4.5	Check-list	211

5 << Concepts avancés

212

5.1	Le DataBinding en détails	214
	DataContext	216
	Interaction avec l'utilisateur	218
5.2	Les Styles et ControlTemplates	220
	Style	220
	ControlTemplate	222
5.3	Créer un UserControl	225
	UserControl ClickMe	226
	UserControl Ranking	227
	Les DependencyProperties	228
	Création de l'UserControl Ranking	229
	MediaElement	237
5.4	Les contrôles de la librairie System.Windows.Controls	246
	Calendar	247
	DatePicker	249
	GridSplitter	251
	TabControl et TabItem	252
5.5	Le contrôle DataGrid	253
	DataGrid non auto généré	255
5.6	Les contrôles Silverlight Toolkit de CodePlex	269
5.7	Check-list	271

6 << Découvrir Deepzoom 272

6.1	Introduction à Deepzoom	274
6.2	Fonctionnement de Deepzoom	275
6.3	Deepzoom par l'exemple	279
	MouseWheelHelper.cs	285
6.4	Deepzoom et Virtual Earth	288
6.5	Check-list	289

7 << Annexes 290

7.1	Silverlight et les langages dynamiques	292
	Silverlight et IronPython	292
	Silverlight et IronRuby	297
	Check-list	299
7.2	Introduction au C#	299
	Déclaration d'une variable de type primitif	300
	Règles de nommage	300
	Déclaration d'une variable de type de classe	301
	Fonctionnement par référence des types de classe	301
	Portée des variables	302
	Utilisation des propriétés de classe	303
	Utilisation des méthodes de classe	303
	Structure d'un programme C# (Partie 1)	303
	Définir un type de classe	304
	Définir une nouvelle méthode	305
	Ajouter une méthode à une classe	306
	Structure d'un programme C# (Partie 2)	306
	Exemple d'une application de gestion de données	307
	Conclusion	311
	Check-list	311
7.3	Webographie	311
	Visual Studio 2008	312
	Silverlight	312
	Le Framework .NET	313

8 << Index 315

Remerciements

Nous remercions les personnes qui nous ont aidé à la réalisation de ce livre et en particulier Stéphanie Fanara qui s'est proposée comme première relectrice de l'ouvrage.

Bargelot (Loïc Bar et Simon Boigelot)

Préface

Définition de Silverlight

Microsoft Silverlight est une plateforme de développement d'applications web de haute qualité (*RIA : rich Internet application*).

Cette plateforme est basée sur la plateforme *.NET*, ce qui en fait la plateforme la plus rapide disponible sur Internet actuellement.

Malgré leur développement en *.NET*, les applications *Silverlight* sont portables. Autant sur Linux, Solaris, Windows et Mac Os que sur certains mobiles.

Pour ce faire, le client web doit installer sur sa machine un sous-ensemble de la plateforme *.NET* : le plugin *Silverlight*.

Ce sous-ensemble contient tout ce qui est nécessaire au fonctionnement de petites applications.

Les prérequis pour débiter

D'un point de vue technique, pour programmer des applications *Silverlight*, le minimum vital est :

- la plateforme *.NET 3.5* ;
- un exemplaire de *Visual Studio 2008*. (La version gratuite de *Visual Studio C# Express* suffit amplement.) ;
- le *Service pack 1 pour Visual Studio 2008* ;
- les *Silverlight Tools pour Visual Studio 2008 (SDK)*.

Tous ces programmes sont disponibles en téléchargement sur le site <http://www.silverlight.net/GetStarted>.

Il est pourtant conseillé d'ajouter à cette liste :

- une version de *Expression Blend 2* ;
- le *Silverlight ToolKit de CodePlex*.

D'un point de vue des connaissances, il est utile d'avoir de notions relatives à :

- la programmation orientée objets ;
- le langage C# ;
- le langage XML.

Pour accéder à une approche de la programmation .NET grâce à C#, reportez-vous à l'annexe 2, Introduction au C#.

Présentation du Microsoft Framework .NET

Le Framework *.Net*, ou plateforme *.NET* en français, est un ensemble composé des éléments suivants :

- une machine virtuelle capable d'exécuter un code intermédiaire propre à la plateforme ;
- un ensemble de langages parmi lesquels *C#*, *VB.NET*, *ASP.NET*, *PHP.NET*, *IronPython*, *IronRuby*, etc. ;
- un ensemble de bibliothèques fournissant un grand nombre d'*API* préprogrammées.

La version actuelle de la plateforme est 3.5. Au début, prévue pour la création d'applications de bureautiques et de serveurs, cette plateforme s'est peu à peu développée pour les sites web et les mobiles.

De nombreux paradigmes de programmation accompagnent ce Framework. Le plus connu étant celui de *la séparation du code d'interface et du code de la logique applicative*. La majorité des langages implémentés par la plateforme divise le code d'une application en deux fichiers séparés.

Ce paradigme permet principalement une meilleure collaboration entre programmeurs, designers et intégrateurs.

Parmi les bibliothèques fournies, vous trouverez tout ce qui concerne les accès fichier, les protocoles de communication, la gestion des données, les connexions aux bases de données, etc.

Chaque nouvelle version du Framework ajoute une couche d'abstraction qui rapproche le langage programmé du langage humain.

Ainsi la version 3.5 a ajouté les requêtes *LINQ* permettant de simplifier grandement le maniement des données.

La version 4, en beta actuellement, simplifiera principalement l'accès aux langages dynamiques. Entre autres, une bibliothèque nommée *DLR (Dynamic Language Runtime)* va permettre aux développeurs de créer de nouveaux langages ou de migrer des langages existant sur la plateforme .NET.

Fonctionnement de Silverlight

Pour faire tourner une application Silverlight dans le navigateur web d'un utilisateur, cet utilisateur doit préalablement installer un plugin.

Ce plugin contient une machine virtuelle prête à interpréter le code *XAML* et certaines bibliothèques de la plateforme *.NET*.

Une fois installé, et lors de la visite d'une page *HTML* contenant un contrôle Silverlight, le plugin va en télécharger le contenu.

Ce contenu est un fichier *XAP*. Un fichier *XAP* est un fichier *ZIP* contenant tout les documents nécessaires au bon fonctionnement d'une application Silverlight.

Ces documents sont principalement :

- les différents fichiers *XAML*, décrivant les interfaces de l'application ;
- les différents fichiers *C#* (ou autre langage) décrivant la logique applicative ;
- les éventuels *médias* (images, vidéos, musique, polices de caractères) ;
- des *bibliothèques .NET* non présentes dans le plugin Silverlight de base.

L'application va ensuite être exécutée dans une zone mémoire de l'ordinateur client. Cette zone est une zone sécurisée du nom de *SandBox* (bac à sable).

Le bac à sable empêche l'application Silverlight de nuire à la machine hôte. Par exemple, en limitant l'accès aux fichiers.

Une application Silverlight garde tout de même la possibilité d'écrire et de lire des fichiers sur l'ordinateur de l'utilisateur. Cependant, cette fonctionnalité est limitée en taille et demande l'approbation préalable de l'utilisateur.

Ce même principe s'applique, par exemple, aux accès aux services web.

La version *1.0* de Silverlight permettait l'utilisation :

- des graphique 2D ;
- d'animations ;
- de média ;
- d'un code JavaScript comme code applicatif.

La version *1.1* a permis l'utilisation de code *C#* ainsi que de quelques autres langages du Framework.

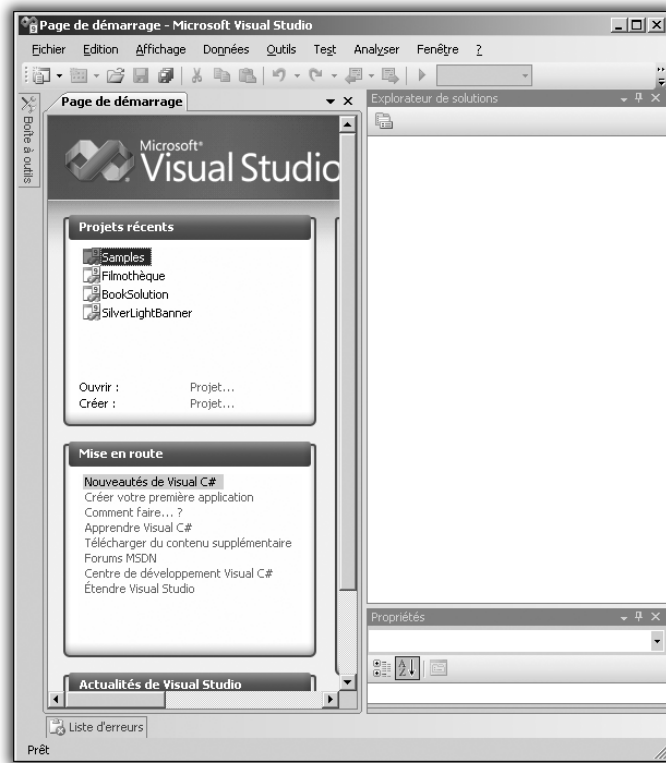
La version 2.0, dernière en date, présentée dans ce livre, a ajouté de nombreuses simplifications dans l'écriture du code *XAML*, de nouveaux *contrôles utilisateur* et une version encore plus rapide du runtime.

Prise en main de Visual Studio 2008

Visual Studio est une application vous aidant à développer. Elle gère des *solutions*. Une solution est un ensemble de projets travaillant en parallèle pour résoudre un problème donné.

Lors du premier démarrage de *Visual Studio*, le programme vous demande quel type de profil vous souhaitez utiliser.

Ce livre est conçu sous le profil *Visual C# Développeur*. Prenez en compte le fait que les raccourcis clavier ainsi que le formatage du texte peuvent changer en fonction du profil choisi.

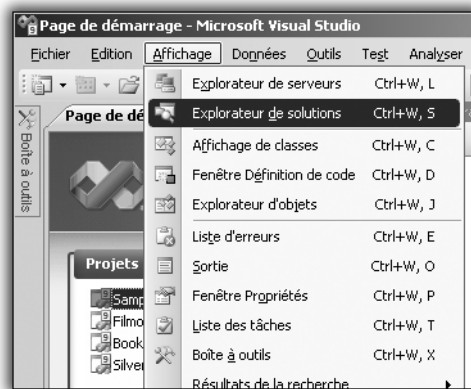


► Fig. 1 : Écran de démarrage de Visual Studio

Le premier écran ouvert dans Visual Studio est la *page de démarrage*. Cette page vous donne un accès rapide aux dernières solutions ouvertes, aux actions de création et d'ouverture d'autres solutions ainsi qu'au dernier post intéressant du Web.

La partie en haut se nomme la barre de menu, la partie placée à l'extrémité droite contient deux outils : l'*Explorateur de solution* et l'*Éditeur de propriétés*.

Si ces outils ne sont pas visibles sous votre configuration actuelle de *Visual Studio*, vous pouvez les activer via le menu **Affichage** de la Barre de menu.

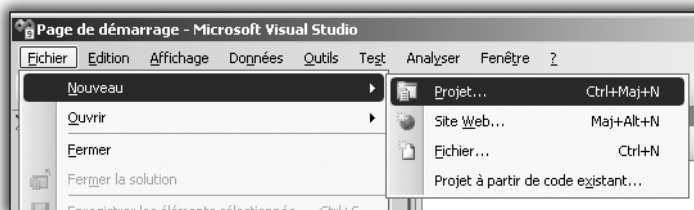


► Fig. 2 :
Menu Affichage

Créer une nouvelle solution Silverlight

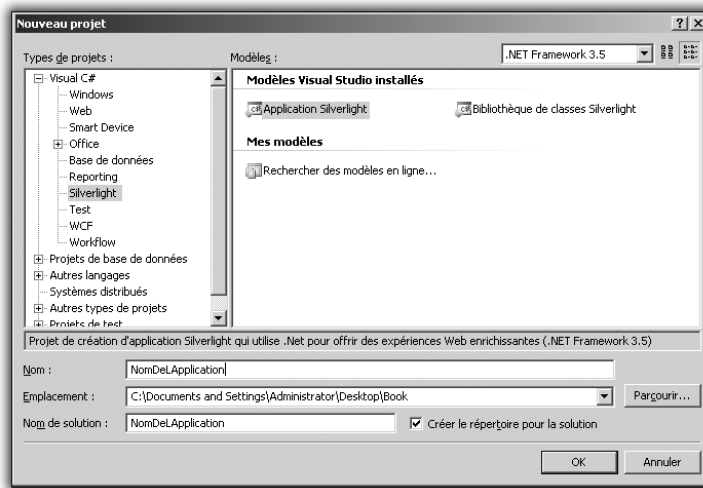
Pour créer une nouvelle application Silverlight :

- 1 Sous le menu **Fichier**, ouvrez le sous-menu **Nouveau** et choisissez l'action **Projet**.



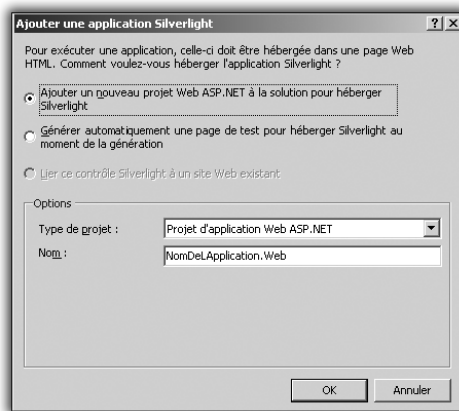
► Fig. 3 : Nouveau Projet

- 2 Dans la boîte de dialogue qui s'affiche, naviguez vers le type de projet **Visual C#** → **Silverlight**.



► Fig. 4 : Boîte de dialogue Nouveau projet

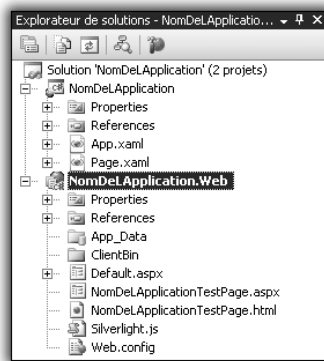
- 3 Sélectionnez le modèle *Application Silverlight*.
- 4 Donnez un nom à votre application et cliquez sur OK.



► Fig. 5 :
Boîte de dialogue type
de débogage

- 5 Une nouvelle boîte de dialogue s'ouvre. Elle vous demande si vous désirez générer un projet contenant un site web pour héberger votre application Silverlight immédiatement ou si vous préférez que *Visual Studio* se charge de cette tâche à chaque démarrage d'une session de débogage de votre application.
- 6 Ajoutez un nouveau projet *ASP.NET* à la solution pour héberger Silverlight.

Votre solution est créée et contient déjà de nombreux fichiers.



► Fig. 6 :
Arborescence de la solution

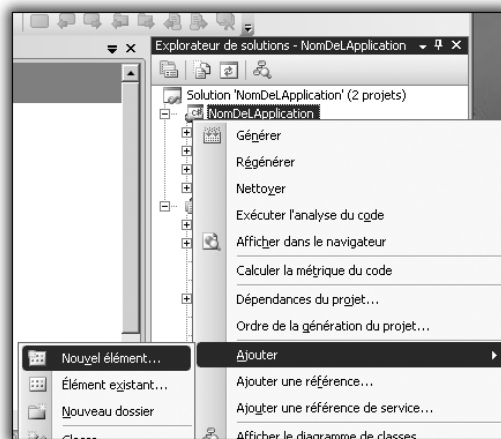
Les trois seuls fichiers qui nous importent pour le moment sont :

- *Page.xaml* contenant la définition de l'interface de votre application ;
- *Page.xaml.cs* qui détient la définition du code application de votre application ;
- *NomDeLApplicationTestPage.aspx*, page ASP.NET hôte de votre *SilverLight*.

Ajouter un nouveau fichier au projet

Pour ajouter un nouveau fichier à un projet sous *Visual Studio* :

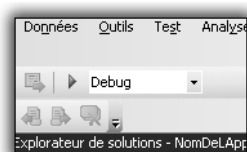
- 1 Cliquez du bouton droit sur le nom du projet dans l'Explorateur de la solution.
- 2 Cliquez sur le menu **Ajouter/Nouvel élément**.



► Fig. 7 :
Ajouter un nouvel
élément à un projet

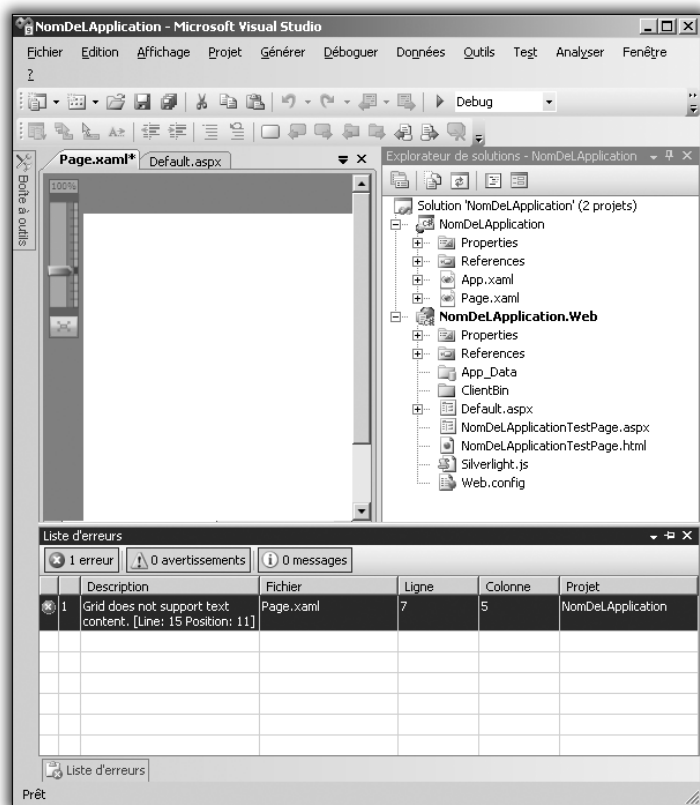
Compiler un projet

- 1 Pour déboguer un projet *Silverlight*, cliquez sur le bouton **Play** situé dans la barre d'outils juste au-dessous de la Barre de menu (ou appuyez sur **F5**).



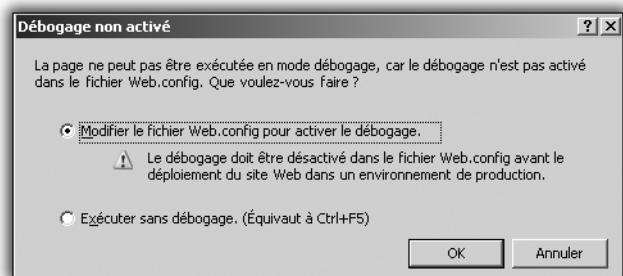
► Fig. 8 :
Bouton Déboguer

Après compilation réussie, votre projet sera lancé dans une page *Internet Explorer*. Si une erreur survient, la liste d'erreurs présente en bas de *Visual Studio* vous en informera.



► Fig. 9 : Liste d'erreurs

Lors de la première exécution de votre application, il est probable que vous trouviez la boîte de dialogue **Débogage non activé**.



► Fig. 10 :
Débogage non activé

- 2 Choisissez l'option *Modifier le fichier Web.config pour activer le débogage* et cliquez sur OK sans vous en soucier davantage.

Ma première application Silverlight : HelloWorld

Quoi de plus beau qu'un HelloWorld !

- 1 Après avoir créé une solution Visual Studio 2008 contenant un projet Silverlight et un projet ASP.NET destinés aux tests, ouvrez le fichier *Page.xaml*.

 Page.xaml

```
<UserControl x:Class="HelloWorld.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">

        </Grid>
    </UserControl>
```

Du code y est déjà présent, il s'agit de code *XAML*, nous étudierons ce langage en détails aux chapitres 2 et 5.

2 En attendant, ajoutez simplement quelques lignes de code :



Page.xaml transformée en HelloWorld

```
<UserControl x:Class="HelloWorld.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <TextBlock Text="HelloWorld" FontFamily="Verena"
            FontSize="70"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"/>
    </Grid>
</UserControl>
```

3 Lancez l'application.



► Fig. 11 :
HelloWorld

Présentation du contenu de ce livre

Ce livre traite de Silverlight 2.0 et non des sujets connexes ; nous nous efforcerons de vous fournir le maximum d'informations pour satisfaire votre curiosité et votre compréhension. Cependant, impossible de résumer dans un seul livre la somme énorme des connaissances et des savoirs requis pour devenir un professionnel du développement d'applications Silverlight.

Notre voyage commencera par un dégrossissement de la forêt impénétrable que compose un fichier *XAML*.

Une fois ce dégrossissement terminé, nous pourrons obtenir un court aperçu de la suite *Expression*. Cette suite est un studio complet destiné au design et à la création artistique.

Nous étudierons ensuite différentes *méthodes d'accès aux données*, quelle qu'en soit leur provenance. Il sera alors temps de revenir sur nos bases de *XAML* pour en apprendre davantage.

Pour finir, nous verrons comment intégrer au mieux Silverlight au sein d'une application *ASP.NET*.



1.1 Introduction	24
1.2 Les bases de XAML	24
1.3 Les éléments de structure	26
1.4 Les éléments de contenu	35
1.5 Les événements et leur traitement	38
1.6 Les éléments d'interactions	40
1.7 Autres éléments utiles	51
1.8 Première approche du DataBinding	55
1.9 Colorez votre application grâce aux Brushes et aux Gradients	73
1.10 Animez votre application grâce aux StoryBoard	79
1.11 Check-List	95

Le langage XAML

Lors de la création du *Framework 3.0*, Microsoft a mis un point d'honneur à simplifier l'interaction entre les développeurs, les intégrateurs et les designers lors du processus de développement d'une application. *XAML (Extensible Application Markup Language)* est la clé de voûte de cette simplification concernant *Windows Presentation Foundation*.

1.1 Introduction

L'interface d'une application y est définie, ou dessinée, sous la forme d'un arbre *XML*. On retrouve dans cette méthode de développement une approche très proche de ce qui se fait depuis longtemps en *ASP.NET*, avec d'un côté le code *HTML*, agrémenté de contrôle *ASP.NET*, et de l'autre, un code de logique applicative contenant la logique de l'application.

De nombreux outils tels que *Expression Blend* vous permettent de faire abstraction du *XAML* en vous proposant une interface interactive de définition de votre application.

Cependant, en tant que développeur, une bonne connaissance du *XAML* vous aidera souvent. En effet, certaines opportunités offertes par *Silverlight* sont bien plus dépendantes du *XAML* que du code applicatif. Le *binding* en est un bon exemple.

1.2 Les bases de XAML

Héritage de XML

Le langage *XAML* est basé sur le langage *XML*.

Cela lui donne déjà de nombreuses bases :

- Un *document* contient toujours un unique élément appelé *élément racine*.
- Un *élément* est une suite de caractères respectant une nomenclature précise.
- Un *élément* peut contenir des *attributs* et/ou des *éléments enfants*.

Nomenclatures d'un élément *XML* sans enfants :

- Le premier caractère d'un élément est toujours <.
- Le nom de l'élément suit directement.
- Viennent ensuite les différents attributs sous la forme NomDeLattribut=« valeur ».
- Les derniers caractères de l'élément sont />.

On obtient donc : <Nom [Attribut1=« valeur » [Attribut2=« valeur » [...]]]/>.

Lorsque l'élément a des enfants, la suite de caractères fermants /> est remplacée par >[enfants]</NomDeLélément>.

Adaptation de XAML

En *XAML*, le nom de l'élément est remplacé par le nom de la classe du contrôle utilisateur qu'il représente. Ainsi, on peut ajouter un bouton à notre interface de la manière suivante :

```
<Button/>
```

Ce bouton peut avoir des attributs, tels qu'une taille et un nom (ce qui permettra de le retrouver plus tard dans le code de la logique applicative).

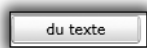
```
<Button Width="100" Name="Button1"/>
```

En *XAML*, tout attribut peut être transformé en enfant. Cette particularité nous sera très utile pour déclarer des attributs structurés :

```
<Button>
  <Button.Width>100</Button.Width>
  <Button.Name>Button1</Button.Name>
</Button>
```

Tout contrôle utilisateur est capable de contenir d'autres contrôles utilisateur. Ainsi, l'emploi le plus fréquent du contenu d'un bouton est du texte :

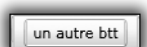
```
<Button Content="du texte">
  <Button.Width>100</Button.Width>
  <Button.Name>Button1</Button.Name>
</Button>
```



► Fig. 1.1 :
Bouton contenant du texte

Mais il peut en être autrement. Par exemple, voici un bouton contenant un autre bouton :

```
<Button Width="100" Name="Button1">
  <Button Width="80" Name="Button2" Content="un autre btt"/>
</Button>
```



► Fig. 1.2 :
Bouton contenant un autre bouton

Attention, la majorité des contrôles utilisateur n'accepte qu'un enfant. Il est donc impossible d'écrire directement :

```
<Button Width="100" Name="Button1">
  <Button Width="80" Name="Button2" Content="un autre btt"/>
  <Button Width="80" Name="Button3" Content="un autre btt"/>
</Button>
```

Pour obtenir un résultat de ce genre, il est indispensable d'utiliser un des *éléments de Layout* que nous verrons au chapitre 3, *Créer vos applications avec Expression Studio*.

Il existe trois types d'attributs différents :

- Le premier type se trouve partout ou presque, tels `Width` et `Height` qui permettent de définir la taille d'un élément d'interface.
- Le deuxième type est un attribut qui se retrouve seulement dans un élément. Un attribut du deuxième type ne pourra jamais être utilisé dans un autre élément.
- Le troisième type d'attribut est plus subtil. Un attribut de ce type appartient à un élément mais est utilisé dans un autre. Un bon exemple est l'attribut `Top` de l'élément `Canvas` :



Attribut de troisième type

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="400" Height="200">
  <Canvas x:Name="LayoutRoot" Background="White">
    <Button Canvas.Top="50"/>
  </Canvas>
</UserControl>
```

1.3 Les éléments de structure

Grid

Lors de la création de votre première application *Silverlight*, le premier élément que vous rencontrez est une *Grid*.

La *Grid*, grille en français, fait partie des éléments structurants (Dit de *Layout*) de votre application.

En effet, si en *WinForm* les différents éléments de vos applications étaient organisés les uns par rapport aux autres majoritairement grâce à leurs positions relatives, en *XAML*, des éléments dits de *Layout* se chargent de cette organisation. Le plus commun de ces éléments est la grille.

Une grille peut être comparée à un tableau composé de lignes et de colonnes. Chaque élément qu'elle contient doit lui spécifier sa position.

Le comportement de base d'une grille stipule qu'elle est composée d'une cellule unique. Cette cellule tentera de donner à son contenu la taille maximale possible :

⚙ Comportement d'une Grid sans cellules

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
    <Button Content="Hello World"/>
  </Grid>
</UserControl>
```



► Fig. 1.3 :
Comportement d'une
Grid sans cellules

Il est pourtant possible de stipuler à une grille le nombre de colonnes qu'elle doit afficher. Sans autres informations, chaque colonne aura une taille égale aux autres. Il est maintenant devenu indispensable, pour le bouton contenu dans la grille, de préciser sa position :

⚙ Comportement d'une Grid avec ColumnDefinitions

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
    <Grid.ColumnDefinitions>
```

```

        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Button Grid.Column="1" Content="Hello World"/>
</Grid>
</UserControl>

```

Ajouter des lignes à cette grille se fait de la même manière :



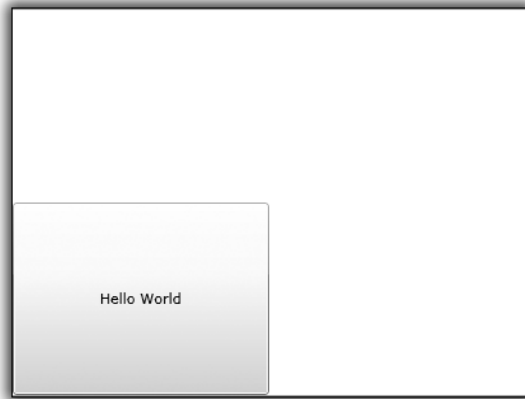
► Fig. 1.4 :
Comportement d'une
Grid avec
ColumnDefinitions

⚙ Comportement d'une Grid avec RowDefinitions

```

<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Button Grid.Column="0" Grid.Row="1" Content="Hello World"/>
    </Grid>
</UserControl>

```



► Fig. 1.5 :
Comportement d'une
Grid avec
RowDefinitions

Trois options s'offrent à nous pour forcer les différentes lignes ou colonnes à adopter des tailles différentes :

- en leur donnant une taille en pixels (Width=100) ;
- en leur demandant de prendre la taille de leur contenu (Width=Auto) ;
- en leur allouant une partie de l'espace restant (Width=*).



Variation de tailles sur les ColumnDefinitions et les RowDefinitions

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="100"/>
      <ColumnDefinition Width="Auto"/>
      <ColumnDefinition Width="*/>
      <ColumnDefinition Width="2*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="100"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*/>
      <RowDefinition Height="2*"/>
    </Grid.RowDefinitions>
    <Button Grid.Column="0" Grid.Row="0" Content="0,0"/>
    <Button Grid.Column="1" Grid.Row="0" Content="1,0"/>
    <Button Grid.Column="2" Grid.Row="0" Content="2,0"/>
    <Button Grid.Column="3" Grid.Row="0" Content="3,0"/>
  </Grid>
</UserControl>
```

```

<Button Grid.Column="0" Grid.Row="1" Content="0,1"/>
<Button Grid.Column="1" Grid.Row="1" Content="1,1"/>
<Button Grid.Column="2" Grid.Row="1" Content="2,1"/>
<Button Grid.Column="3" Grid.Row="1" Content="3,1"/>
<Button Grid.Column="0" Grid.Row="2" Content="0,2"/>
<Button Grid.Column="1" Grid.Row="2" Content="1,2"/>
<Button Grid.Column="2" Grid.Row="2" Content="2,2"/>
<Button Grid.Column="3" Grid.Row="2" Content="3,2"/>
<Button Grid.Column="0" Grid.Row="3" Content="0,3"/>
<Button Grid.Column="1" Grid.Row="3" Content="1,3"/>
<Button Grid.Column="2" Grid.Row="3" Content="2,3"/>
<Button Grid.Column="3" Grid.Row="3" Content="3,3"/>
</Grid>
</UserControl>

```

0,0	1,0	2,0	3,0
0,1	1,1	2,1	3,1
0,2	1,2	2,2	3,2
0,3	1,3	2,3	3,3

► Fig. 1.6 :
Variation de tailles sur les
ColumnDefinitions et les
RowDefinitions

StackPanel

Le deuxième élément de la famille des Layout est le *StackPanel*. Un *StackPanel* imbrique ses différents enfants les uns au-dessous des autres :



StackPanel

```

<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="400" Height="100">
  <Grid x:Name="LayoutRoot" Background="White">
    <StackPanel>
      <Button Content="Hello"/>
      <Button Content="World"/>
      <Button Content="!!!" />
    </StackPanel>
  </Grid>
</UserControl>

```

```

    </StackPanel>
  </Grid>
</UserControl>

```



► Fig. 1.7 :
StackPanel à orientation
verticale

Les éléments fournis dans la plateforme *Silverlight* de base ne proposent pas de *WrapPanel* tel qu'on en trouve en *WPF*. Il existe deux solutions pour pallier ce problème :

- utiliser le *Silverlight Toolkit de CodePlex* que nous évoquerons plus tard ;

Pour obtenir plus de renseignements sur le Silverlight Toolkit de CodePlex, reportez-vous au chapitre 5, Silverlight et ASP.NET.

- forcer un *StackPanel* à afficher ses enfants les uns à côté des autres.



StackPanel à orientation horizontale

```

<StackPanel Orientation="Horizontal">
  <Button Content="Hello"/>
  <Button Content="World"/>
  <Button Content="!!!"/>
</StackPanel>

```



► Fig. 1.8 :
StackPanel à orientation
horizontale

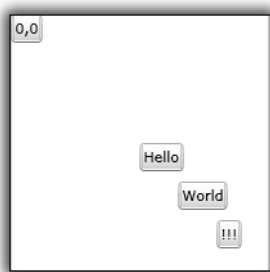
Canvas

Le dernier élément de la famille des *Layout* présenté dans ce livre est le *Canvas*. Son utilisation est la même qu'à l'époque des *WinForm*. En effet, le *Canvas*, contrairement aux autres *Layout*, délègue le positionnement de ses enfants à eux-mêmes.

Ce positionnement est relatif à leurs distances par rapport au dessus et à la gauche du *Canvas* :

Canvas

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="200">
  <Canvas x:Name="LayoutRoot" Background="White" >
    <Button Canvas.Top="0" Canvas.Left="0" Content="0,0"/>
    <Button Canvas.Top="100" Canvas.Left="100" Content="Hello"/>
    <Button Canvas.Top="130" Canvas.Left="130" Content="World"/>
    <Button Canvas.Top="160" Canvas.Left="160" Content="!!!"/>
  </Canvas>
</UserControl>
```



► Fig. 1.9 :
Exemple de Canvas

ScrollViewer

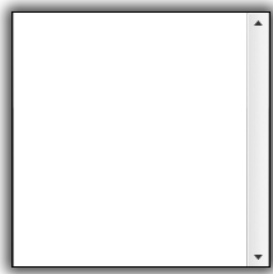
Bien que le *ScrollViewer* ne soit pas à proprement parler un élément de Layout, il permet tout de même de structurer l'interface.

C'est lui qui vous offre l'opportunité d'ajouter des ascenseurs à votre application, qu'ils soient horizontaux ou verticaux.

Sans configuration, seul l'ascenseur vertical est visible :

ScrollViewer sans configuration

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="200">
  <ScrollViewer x:Name="LayoutRoot" Background="White" >
  </ScrollViewer>
</UserControl>
```

► Fig. 1.10 :
ScrollViewer sans configuration

Les attributs `HorizontalScrollBarVisibility` et `VerticalScrollBarVisibility` permettent de modifier cette configuration. Ils acceptent 4 valeurs différentes :

- **Auto** : l'ascenseur est visible uniquement si le contenu du *ScrollViewer* dépasse la taille du *ScrollViewer*.
- **Disabled** : l'ascenseur est visible mais inutilisable par l'utilisateur.
- **Hidden** : l'ascenseur est caché.
- **Visible** : l'ascenseur est toujours visible. Si la taille du contenu dépasse la taille du *ScrollViewer*, l'ascenseur pourra être employé par l'utilisateur.

⚙ ScrollViewer avec configuration

```
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="200" Height="200">
    <ScrollViewer x:Name="LayoutRoot" Background="White"
        HorizontalScrollBarVisibility="Auto"
        VerticalScrollBarVisibility="Auto">

    </ScrollViewer>
</UserControl>
```

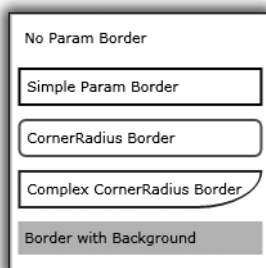
Border

Au même titre que le *ScrollViewer*, l'élément *Border* n'est pas un élément de Layout. Cependant, son utilisation permet de rendre plus compréhensible la structure d'une interface pour l'utilisateur.

Un *Border* est seulement une ligne qui entoure l'élément qu'il contient. Ses différents attributs lui donnent une flexibilité incomparable et font de lui un atout majeur :

⚙️ Différents exemples de Border

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="200">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <Border Margin="5">
      <TextBlock Text="No Param Border" Margin="5"/>
    </Border>
    <Border Margin="5"
      BorderBrush="Black"
      BorderThickness="2">
      <TextBlock Text="Simple Param Border" Margin="5"/>
    </Border>
    <Border Margin="5"
      BorderBrush="Green"
      BorderThickness="2"
      CornerRadius="5">
      <TextBlock Text="CornerRadius Border" Margin="5"/>
    </Border>
    <Border Margin="5"
      BorderBrush="Blue"
      BorderThickness="2"
      CornerRadius="0,0,50,0">
      <TextBlock Text="Complex CornerRadius Border" Margin="5"/>
    </Border>
    <Border Margin="5"
      BorderBrush="Black"
      Background="Aqua">
      <TextBlock Text="Border with Background" Margin="5"/>
    </Border>
  </StackPanel>
</UserControl>
```



► Fig. 1.11 :
Différents exemples de Border

Les attributs Margin et CornerRadius

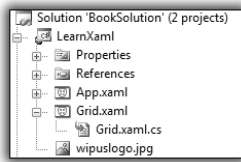
Remarquez l'utilisation de l'attribut **Margin**. Ce dernier, de type **1**, est disponible dans la majorité des éléments d'interfaces et permet de les espacer les uns des autres. Comme le **CornerRadius** dans sa version complexe, le **Margin** peut être défini avec 4 valeurs (**Top**, **Right**, **Bottom**, **Left**).

1.4 Les éléments de contenu

Images

L'élément **Image** est enfantin à utiliser. Pour ajouter une image à votre application, il suffit d'en indiquer le chemin relatif par rapport au nœud racine de votre application ou d'en indiquer une URI absolue.

Par exemple, pour afficher un fichier *wipuslogo.jpg* faisant partie de votre projet *Silverlight* :



► Fig. 1.12 :
Structure de la solution

Image avec Source relative au projet

```
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="400" Height="200">
    <Grid x:Name="LayoutRoot" Background="White">
        <Image Source="wipuslogo.jpg"/>
    </Grid>
</UserControl>
```

Ou encore, afficher une image récupérée à partir d'un site web :



► Fig. 1.13 :
Image avec source
relative au projet

⚙️ Image avec Source absolue (URI)

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="400" Height="200">
  <Grid x:Name="LayoutRoot" Background="White">
    <Image Source="http://www.wipus.com/WipusCloud.jpg" />
  </Grid>
</UserControl>
```



► Fig. 1.14 :
Image avec source
absolue (URI)

Un des attributs les plus importants de l'élément Image est l'attribut Stretch. Il permet de définir comment l'élément va interagir avec l'image pour l'afficher, s'il va la redimensionner, la couper, en conservant les proportions d'origine ou non.

TextBlock

Le TextBlock est la zone de texte la plus communément utilisée. Elle permet de choisir toutes les caractéristiques habituelles d'un texte, de sa famille à sa taille, en passant par sa couleur :

⚙️ TextBlock

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="200">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <TextBlock Text="texte blablabla"
      TextAlignment="Center"
      TextWrapping="NoWrap"
      FontFamily="Arial"
      FontSize="24"
      FontStyle="Italic">
```

```

        FontWeight="Bold"
        Foreground="Blue"/>
    <TextBlock Text="texte 2"
        TextAlignment="Left"
        TextWrapping="NoWrap"
        FontFamily="Verena"
        FontSize="13"
        FontWeight="Bold"
        Foreground="Red"/>
    <TextBlock Text="texte 3"
        TextAlignment="Right"
        TextWrapping="NoWrap"
        FontFamily="Verena"
        FontSize="17"
        FontWeight="Bold"
        Foreground="Magenta"/>
</StackPanel>
</UserControl>

```



► Fig. 1.15 :
Exemple de TextBlocks

Remarquez l'attribut TextWrapping. Il permet de passer automatiquement à la ligne lorsque sa valeur est Wrap. Le cas échéant, si la longueur du texte dépasse la taille du TextBlock, le surplus sera coupé.

ProgressBar

Comme son nom l'indique, la ProgressBar sert à indiquer l'état d'une progression, il s'agit de la même ProgressBar que celle utilisée par Windows lors d'une copie. Son attribut Value indique l'état actuel de la progression qui peut varier de Minimum à Maximum :



ProgressBar

```

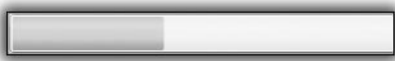
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="300" Height="30">

```

```

<StackPanel x:Name="LayoutRoot" Background="White">
    <ProgressBar Name="ProgressBar1"
        Height="30"
        Maximum="100"
        Value="40"
        Minimum="0"
        Foreground="Green"
    />
</StackPanel>
</UserControl>

```



► Fig. 1.16 :
ProgressBar

1.5 Les événements et leur traitement

Avant de continuer, il est important de comprendre comment fonctionne le traitement des événements en *Silverlight*, comment interagir entre le code *XAML* et le code *C#* de l'application.

Dès qu'un élément, tel qu'un bouton, est ajouté au code *XAML*, et pour peu que cet élément contienne un attribut *Name*, il devient accessible dans le code *C#*.

Ainsi pour atteindre le bouton de *Name* *Button1* dans le code *C#* d'une application, il suffit d'écrire son nom. Tout attribut en *XAML* devient alors une propriété en *C#* :



Code XAML

```

<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="300" Height="30">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button Name="Boutton1" Content="un bouton"/>
    </Grid>
</UserControl>

```



Code C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;

```

```

using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace LearnXAML
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();

            string temp = "" + Boutton1.Content;
        }
    }
}

```

Pour ajouter l'événement Clic sur ce bouton, indiquez-le comme attribut dans le code *XAML* et ajoutez la fonction relative dans le code *C#*. Généralement, cette opération est automatique :



Modification du code XAML

```
<Button Name="Boutton1" Content="un bouton" Clic="Boutton1_Clic"/>
```



Modification du code C#

```

using ...

namespace LearnXAML
{
    public partial class Page : UserControl
    {
        public Page()
        {...}

        private void Boutton1_Clic(object sender, RoutedEventArgs e)
        {
            Boutton1.Content = "Merci";
        }
    }
}

```



► Fig. 1.17 : Événement Clic sur bouton

Les événements relatifs aux entrées utilisateur sont partagés par la majorité des éléments d'interface.

Ainsi `MouseEnter`, `MouseMove`, `MouseLeftButtonDown`, `MouseLeftButtonUp`, `MouseLeave`, `KeyUp` et `KeyDown` se retrouvent presque partout.

Absence de clic droit en Silverlight

En *Silverlight*, il n'y a pas d'événements liés au clic droit de la souris.

1.6 Les éléments d'interactions

Button

Le bouton est l'élément d'interaction par excellence que nous avons déjà largement utilisé.

Voici, pour rappel, son fonctionnement :

Code XAML d'un bouton

```
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="100" Height="30">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Text" Clic="Boutton1_Clic"/>
    </Grid>
</UserControl>
```

Le texte d'un bouton est défini dans son attribut `Content`. En effet, ce contenu peut être autre chose que du texte. Par exemple, une image :

Bouton avec une image comme contenu

```
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML">
```



```

Width="100" Height="100">
  <Grid x:Name="LayoutRoot" Background="White">
    <Button Clic="Boutton1_Clic">
      <Image Source="wipuslogo.jpg" Stretch="UniformToFill"/>
    </Button>
  </Grid>
</UserControl>

```



► Fig. 1.18 :
Bouton avec une image comme contenu

Ou quelque chose de plus complexe tel qu'une grille contenant plusieurs images et un autre bouton :

⚙ Bouton à contenu hétéroclite

```

<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="200">
  <Grid x:Name="LayoutRoot" Background="White">
    <Button Clic="Boutton1_Clic">
      <Grid>
        <Grid.ColumnDefinitions>
          <ColumnDefinition/>
          <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
          <RowDefinition/>
          <RowDefinition/>
        </Grid.RowDefinitions>
        <Image Grid.Row="0"
          Grid.Column="1"
          Source="wipuslogo.jpg"
          Stretch="UniformToFill"/>
        <Image Grid.Row="0"
          Grid.Column="0"
          Source="wipuslogo.jpg"
          Stretch="UniformToFill"/>
        <Image Grid.Row="1"
          Grid.Column="1"
          Source="wipuslogo.jpg"

```

```

        Stretch="UniformToFill"/>
        <Button Grid.Row="1"
            Grid.Column="0"
            Content="Un autre"/>
    </Grid>
</Button>
</Grid>
</UserControl>

```



► Fig. 1.19 :
Bouton à contenu hétéroclite

❗ Généricité du principe des poupées suisses

Ce principe s'applique à tous les éléments XAML.

CheckBox

Une *CheckBox* est une boîte à deux ou trois états selon sa configuration. Son état peut être coché, non coché ou éventuellement inconnu.

Cet état est stocké sous forme d'un booléen annulable dans l'attribut `IsChecked`.

L'attribut `IsThreeState` permet de définir si la *CheckBox* peut ou non passer par l'état inconnu :



Exemple de CheckBox

```

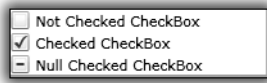
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="200" Height="50">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <CheckBox IsChecked="False" Content="Not Checked CheckBox"/>
    </StackPanel>
</UserControl>

```

```

        <CheckBox IsChecked="True" Content="Checked CheckBox"/>
        <CheckBox IsThreeState="True" IsChecked="{x:Null}"
            Content="Null Checked CheckBox"/>
    </StackPanel>
</UserControl>

```



► Fig. 1.20 :
Exemple de CheckBox

Lorsqu'une *CheckBox* passe d'un état à un autre, les événements *Checked* et *Unchecked* sont déclenchés :

⚙️ Evènements Checked et UnChecked (XAML)

```

<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="200" Height="50">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <CheckBox IsChecked="False" Content="CheckBox"
            Checked="CheckBox_Checked"
            Unchecked="CheckBox_Unchecked"/>
    </StackPanel>
</UserControl>

```

⚙️ Evènements Checked et UnChecked (C#)

```

private void CheckBox_Checked(object sender, RoutedEventArgs e)
{ }

private void CheckBox_Unchecked(object sender, RoutedEventArgs e)
{ }

```

ToggleButton

Un *ToggleButton* est une *CheckBox* sous forme de bouton. Son état peut être enfoncé, non-enfoncé ou éventuellement inconnu.

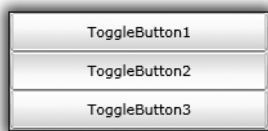
Cet état est stocké sous forme d'un booléen annulable dans l'attribut *IsChecked*.

L'attribut `IsThreeState` permet de définir si le `ToggleButton` peut ou non passer par l'état inconnu :



Exemple de `ToggleButton`

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="90">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <ToggleButton Height="30" Content="ToggleButton1"
      IsChecked="False"/>
    <ToggleButton Height="30" Content="ToggleButton2"
      IsChecked="True"/>
    <ToggleButton Height="30" Content="ToggleButton3"
      IsThreeState="True"
      IsChecked="{x:Null}"/>
  </StackPanel>
</UserControl>
```



► Fig. 1.21 :
Exemple de `ToggleButton`

Lorsque l'état d'un *`ToggleButton`* change, les événements `Checked` et `Unchecked` sont déclenchés :



Evènements d'état d'un `ToggleButton`

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="90">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <ToggleButton Height="30" Content="ToggleButton1"
      IsChecked="False"
      Checked="ToggleButton_Checked"
      Unchecked="ToggleButton_Unchecked"/>
  </StackPanel>
</UserControl>
```

RadioButton

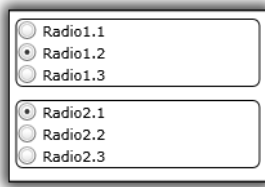
Les *RadioButtons* fonctionnent de la même façon que les *ToggleButtons*. Eux aussi possèdent les attributs `IsChecked`, `IsThreeState`, `Content`, etc.

Cependant, ils offrent une fonctionnalité supplémentaire les rendant intéressants : l'attribut `GroupName`.

Parmi un ensemble de plusieurs *RadioButtons* partageant le même `GroupName`, seul un *RadioButton* pourra avoir son état `IsChecked` à vrai. Si un autre de ces *RadioButtons* passe à l'état `Checked`, automatiquement, le précédent *RadioButton* `Checked` deviendra `Unchecked` :

Exemple de RadioButtons

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="130">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <Border BorderBrush="Black" BorderThickness="1"
      CornerRadius="5" Margin="5">
      <StackPanel>
        <RadioButton GroupName="Group1" Content="Radio1.1"/>
        <RadioButton GroupName="Group1" Content="Radio1.2"
          IsChecked="True"/>
        <RadioButton GroupName="Group1" Content="Radio1.3"/>
      </StackPanel>
    </Border>
    <Border BorderBrush="Black" BorderThickness="1"
      CornerRadius="5" Margin="5">
      <StackPanel>
        <RadioButton GroupName="Group2" Content="Radio2.1"
          IsChecked="True"/>
        <RadioButton GroupName="Group2" Content="Radio2.2"/>
        <RadioButton GroupName="Group2" Content="Radio2.3"/>
      </StackPanel>
    </Border>
  </StackPanel>
</UserControl>
```



► Fig. 1.22 :
Exemple de RadioButtons

De même que pour le *ToggleButton*, les événements *Checked* et *Unchecked* sont déclenchés lorsqu'un *RadioButton* change d'état.

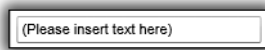
TextBox

Élément de saisie de texte, une *TextBox* offre les mêmes attributs qu'un *TextBlock* :



Exemple de TextBox

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="30">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <TextBox FontFamily="Arial"
      Foreground="Black"
      Text="(Please insert text here)"
      Margin="5"
      TextChanged="TextBox_TextChanged"/>
  </StackPanel>
</UserControl>
```



► Fig. 1.23 :
Exemple de TextBox

L'événement *TextChanged* est déclenché lorsque l'utilisateur a fini d'éditer le texte contenu dans la *TextBox*. Cela étant, hormis pour une validation, le texte entré par l'utilisateur sera utilisé lors d'un autre événement, tel que le clic sur un bouton adjacent.

Pour récupérer le texte contenu dans ce *TextBox* à partir du code *C#*, il suffit d'en utiliser la propriété *Text*.

PasswordBox

Une *PasswordBox* est une *TextBox* n'affichant pas les caractères frappés au clavier mais des caractères de remplacements :

Exemple de PasswordBox

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="30">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <PasswordBox FontFamily="Arial"
      Foreground="Black"
      Password="(Please insert text here)"
      Margin="5"
      PasswordChanged="PasswordBox_PasswordChanged"
      PasswordChar="*" />
  </StackPanel>
</UserControl>
```



► Fig. 1.24 :
Exemple de PasswordBox

Attention, le texte contenu dans une *PasswordBox* porte le nom de *Password* et non de *Text*. Il en va de même pour la propriété dans le code C#.

L'attribut *PasswordChar* donne au développeur l'opportunité de choisir un caractère de remplacement différent de celui de la plateforme d'origine.

ListBox et ListBoxItem

La *ListBox* est un élément composé d'un *StackPanel* et d'un *ScrollViewer*. Mais ce n'est pas tout, elle implémente d'origine une interface lui permettant de gérer une collection d'items sélectionnables :

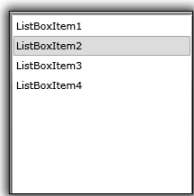
ListBox contenant des ListBoxItems

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="200">
  <Grid x:Name="LayoutRoot" Background="White">
    <ListBox>
```

```

        <ListBoxItem Content="ListBoxItem1"/>
        <ListBoxItem Content="ListBoxItem2"/>
        <ListBoxItem Content="ListBoxItem3"/>
        <ListBoxItem Content="ListBoxItem4"/>
    </ListBox>
</Grid>
</UserControl>

```



► Fig. 1.25 :
ListBox contenant des ListBoxItems

Les éléments utilisés dans cet exemple comme items sont des *ListBoxItems*. Bien que les *ListBoxItems* soient prévus spécialement pour servir d'items dans une *ListBox*, tout autre élément de la plateforme peut les substituer.

Ainsi, il est possible d'avoir une *ListBox* contenant comme items des boutons, des autres *ListBox*, des éléments de *Layout* contenant eux-mêmes d'autres éléments enfants, etc. :



ListBox contenant des éléments hétéroclites

```

<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="200" Height="200">
    <Grid x:Name="LayoutRoot" Background="White">
        <ListBox>
            <Button Content="ListBoxItem1"/>
            <ListBox>
                <ListBoxItem Content="ListBoxItem2.1"/>
                <ListBoxItem Content="ListBoxItem2.2"/>
            </ListBox>
            <StackPanel Orientation="Horizontal">
                <ListBoxItem Content="ListBoxItem3.1"/>
                <Button Content="ListBoxItem3.1"/>
            </StackPanel>
            <ListBoxItem Content="ListBoxItem4"/>
        </ListBox>
    </Grid>
</UserControl>

```




► Fig. 1.26 :
ListBox contenant des éléments hétéroclites

Lorsque la sélection passe d'un élément à un autre, l'événement `SelectedItemChanged` est déclenché.

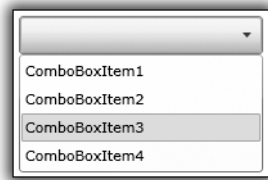
ComboBox et ComboBoxItem

Une *ComboBox* est une *ListBox* présentée sous forme d'une liste déroulante. Son utilisation est donc semblable à celle d'une *ListBox* :



Exemple de ComboBox

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="40">
  <Grid x:Name="LayoutRoot" Background="White">
    <ComboBox Height="30" Margin="5">
      <ComboBoxItem Content="ComboBoxItem1"/>
      <ComboBoxItem Content="ComboBoxItem2"/>
      <ComboBoxItem Content="ComboBoxItem3" IsSelected="True"/>
      <ComboBoxItem Content="ComboBoxItem4"/>
    </ComboBox>
  </Grid>
</UserControl>
```



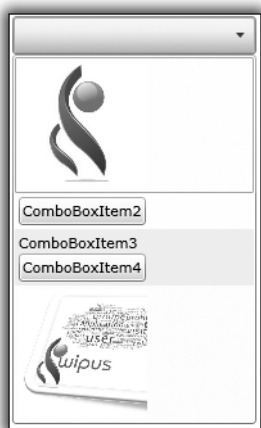
► Fig. 1.27 :
Exemple de ComboBox

Les éléments utilisés dans cet exemple sont des *ComboBoxItems*, mais comme une *ListBox*, les *ComboBox* acceptent n'importe quels éléments comme items.

Il est donc possible, par exemple, de construire une *ComboBox* d'images, de boutons et d'informations diverses réunies :

⚙ Exemple de ComboBox à contenu hétéroclite

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="40">
  <Grid x:Name="LayoutRoot" Background="White">
    <ComboBox Height="30" Margin="5">
      <Image Source="wipuslogo.jpg"
        Height="100" Width="100"
        Stretch="UniformToFill"/>
      <Button Content="ComboBoxItem2"/>
      <StackPanel Orientation="Vertical">
        <TextBlock Text="ComboBoxItem3" />
        <Button Content="ComboBoxItem4"/>
      </StackPanel>
      <Image Source="http://www.wipus.com/WipusCloud.jpg"
        Height="100" Width="100"
        Stretch="UniformToFill"/>
    </ComboBox>
  </Grid>
</UserControl>
```



► Fig. 1.28 :
Exemple de ComboBox à contenu hétéroclite

Lorsque la sélection passe d'un élément à un autre, l'événement `SelectedItemChanged` est déclenché.

Slider

Un *Slider* est un curseur se déplaçant le long d'une ligne pour permettre à l'utilisateur d'indiquer une valeur numérique de façon visuelle. La valeur (*Value*) indiquée par le curseur varie de l'attribut *Minimum* à gauche de la ligne à l'attribut *Maximum* à droite de la ligne :



Exemple de Slider

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="40">
  <Grid x:Name="LayoutRoot" Background="White">
    <Slider Maximum="100"
      Minimum="0"
      Value="40"
      SmallChange="1"
      LargeChange="10"
    />
  </Grid>
</UserControl>
```



► Fig. 1.29 :
Exemple de Slider

Les attributs *SmallChange* et *LargeChange* représentent respectivement la valeur ajoutée ou retirée à la valeur en cours lors d'un déplacement par la souris ou le clavier.

L'événement *ValueChanged* est déclenché lorsque la valeur indiquée par le curseur est modifiée par l'utilisateur.

1.7 Autres éléments utiles

Line

La plateforme n'offre pas que des éléments d'interface mais aussi des éléments graphiques tels que les lignes, les polygones, les rectangles, etc.

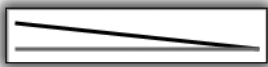
Bien que ces éléments soient plus utiles aux designers qu'aux développeurs, il est intéressant de comprendre leurs fonctionnements :



Exemple de Ligne

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="40">
  <Grid x:Name="LayoutRoot" Background="White">
    <Line X1="5" Y1="10"
          X2="195" Y2="30"
          Stroke="Black"
          StrokeThickness="3"/>

    <Line X1="5" Y1="30"
          X2="195" Y2="30"
          Stroke="Magenta"
          StrokeThickness="3"/>
  </Grid>
</UserControl>
```



► Fig. 1.30 :
Exemple de ligne

Dans cet exemple, deux lignes sont dessinées grâce aux positions relatives (en pixels) de leurs points d'origine et de destination par rapport à l'élément qui les contient.

Ces deux points sont respectivement {X1,Y1} et {X2,Y2}.

Rectangle

Dans le même ordre d'idées, un *rectangle* est défini par ses dimensions. Les attributs RadiusX et RadiusY, quant à eux, permettent d'en arrondir les bords :



Exemple de Rectangle

```
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="200" Height="60">
  <StackPanel x:Name="LayoutRoot" Background="White">
    <Rectangle Fill="Magenta" Margin="5"
      Height="20" Width="100"
      MouseLeftButtonDown="Rectangle_MouseLeftButtonDown"/>
    <Rectangle Fill="Blue" Margin="5"
      Height="20" RadiusX="30" RadiusY="30"/>
  </StackPanel>
</UserControl>
```



► Fig. 1.31 :
Exemple de Rectangle

Interaction utilisateur des éléments graphiques

Souvenez-vous que tout élément graphique garde une possibilité d'interaction avec la souris et/ou le clavier. Dans cet exemple, nous retrouvons l'événement `MouseLeftButtonDown`.

Attention, un rectangle est un élément géométrique et non un élément d'interface ; il n'accepte donc pas de contenu. Le code suivant ne fonctionnera pas :

Rectangle avec contenu

```
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="200" Height="60">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <Rectangle Fill="Magenta" Margin="5"
            Height="20" Width="100"
            MouseLeftButtonDown="Rectangle_MouseLeftButtonDown">
            <Button/>
        </Rectangle>
    </StackPanel>
</UserControl>
```

Pour obtenir un résultat semblable qui fonctionne, utilisez un *Border* avec *Background*.

Popup

En *Silverlight*, un élément *Popup* n'est pas un popup à proprement parler tel qu'on l'entend dans le monde du Web. En effet, il ne s'agit en aucun cas d'une fenêtre supplémentaire qui s'ouvre hors du navigateur dans lequel s'exécute l'application *Silverlight*.

Un popup *Silverlight* n'est rien d'autre qu'un élément d'interface indépendant de la structure solide définie par les éléments de Layout de l'application. Il s'affiche librement au-dessus de tout autre contrôle.

Dans cet exemple, un bouton contrôle l'ouverture d'un popup. Ce popup contient comme éléments enfants un texte et un bouton déclenchant sa fermeture :



Exemple de Popup (XAML)

```
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="200" Height="60">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button Height="20"
            Width="100"
            Clic="OpenPopup"
            Content="Open Popup"/>
        <Popup Name="MyPopup"
            VerticalAlignment="Center"
            HorizontalAlignment="Center">
            <Border BorderBrush="Black"
                BorderThickness="2"
                Background="Beige">
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="Text Popup"
                        Margin="5"/>
                    <Button Margin="5"
                        Clic="ClosePopup"
                        Content="x"/>
                </StackPanel>
            </Border>
        </Popup>
    </Grid>
</UserControl>
```

Ce sont sur les événements Clic des deux boutons que l'on assignera une valeur différente à l'attribut `IsOpen` du popup. Lorsque cet attribut est vrai, le popup est affiché :



Exemple de Popup (C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```

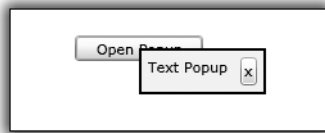
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace LearnXAML
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }

        private void OpenPopup(object sender, RoutedEventArgs e)
        {
            MyPopup.IsOpen = true;
        }

        private void ClosePopup(object sender, RoutedEventArgs e)
        {
            MyPopup.IsOpen = false;
        }
    }
}

```



► Fig. 1.32 : Exemple de Popup

1.8 Première approche du DataBinding

Le *binding* est la méthode de liaison de données entre le code applicatif (code C#) et le code XAML.

Grâce à cette méthode, il devient aisé d'afficher un set de données à l'utilisateur tout en lui proposant une interaction directe avec elles, sans devoir écrire de nombreuses lignes de code.

Pour lier des données à une interface, il nous faut d'abord créer ces données.

Si vous n'êtes pas habitué à la programmation orientée objets, reportez-vous à l'annexe 2 de ce livre, Introduction au C#.

L'application que nous allons réaliser ici se doit de gérer une liste d'étudiants, répartis dans plusieurs cours. Un étudiant est défini par son *Nom*, son *Prénom* et son *Âge* :

**Etudiant.cs**

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace LearnXAML
{
    public class Etudiant
    {
        private string nom;

        public string Nom
        {
            get { return nom; }
            set { nom = value; }
        }

        private string prenom;

        public string Prenom
        {
            get { return prenom; }
            set { prenom = value; }
        }

        private int age;

        public int Age
        {
            get { return age; }
            set { age = value; }
        }

        public Etudiant()
        {
        }
    }
}
```



```
    }
}
```

Un cours, quant à lui, est défini par son *Nom* et comprend une *liste d'étudiants* :

Cours.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Collections.Generic;

namespace LearnXAML
{
    public class Cours
    {
        private string nom;

        public string Nom
        {
            get { return nom; }
            set { nom = value; }
        }

        private List<Etudiant> etudiants;

        public List<Etudiant> Etudiants
        {
            get { return etudiants; }
            set { etudiants = value; }
        }

        public Cours()
        {
        }
    }
}
```

Ajoutons au code applicatif de cette application la déclaration en dur d'une liste de deux cours comprenant chacun quelques étudiants :



Déclaration en dure d'une liste d'étudiant

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace LearnXAML
{
    public partial class Page : UserControl
    {

        private List<Cours> ListeDeCours;

        public Page()
        {
            InitializeComponent();

            #region HardCode ListeDeCours
            ListeDeCours = new List<Cours>()
            {
                new Cours()
                {
                    Nom = "Declamation",
                    Etudiants = new List<Etudiant>()
                    {
                        new Etudiant() {Prenom="Laurane", Nom="D.", Age=19 },
                        new Etudiant() {Prenom="Maï", Nom="H.", Age=18 },
                        new Etudiant() {Prenom="Dovy", Nom="F.", Age=21 }
                    }
                },
                new Cours()
                {
                    Nom = "Sculpture",
                    Etudiants = new List<Etudiant>()
                    {
                        new Etudiant() {Prenom="Adelaïde", Nom="A.", Age=19 },
```

```

        new Etudiant() {Prenom="Klintes", Nom="T.", Age=20 },
        new Etudiant() {Prenom="Kaphilis", Nom="A.", Age=20 },
        new Etudiant() {Prenom="Jade", Nom="G.", Age=14 }
    }
}
};
#endregion HarCode ListeDeCours
}
}
}

```

Dans cette déclaration, nous retrouvons la structure suivante :

- un cours de *déclamation* dont les élèves sont *Laurane*, *Maï* et *Dovy* ;
- un cours de *sculpture* dont les élèves sont *Adelaïde*, *Klintes*, *Kaphilis* et *Jade*.

Du côté de la définition de l'interface, déclarons une `ListBox` :

fichier XAML

```

<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <ListBox Name="CoursListBox"/>
    </Grid>
</UserControl>

```

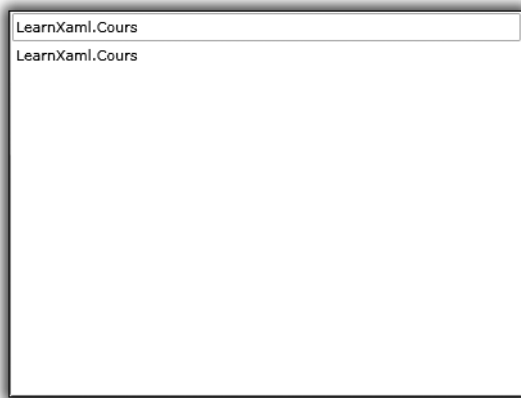
Pour lier la liste de cours à cette *ListBox*, il suffit d'ajouter dans le code applicatif, sous la déclaration en dur :

Ajout au code applicatif

```
CoursListBox.ItemsSource = ListeDeCours;
```

Cette ligne de code ordonne à la *ListBox CoursListBox* d'afficher le contenu de la liste *ListeDeCours*.

Voici le résultat :



► Fig. 1.33 :
DataBinding direct

Bien que cela ne plante pas et que nous détectons déjà 2 items différents dans cette *ListBox*, l'interface obtenue est loin d'être satisfaisante pour un utilisateur final.

Pourquoi ? La *plateforme .Net*, en l'absence d'informations lui dictant comment afficher un objet, en appelle à la méthode `ToString`. Cette méthode, présente intrinsèquement dans chaque objet, retourne une chaîne de caractères comprenant le nom de la classe de cet objet.

Le premier contournement de ce problème est donc la surcharge de cette méthode :



Ajout à Cours.cs

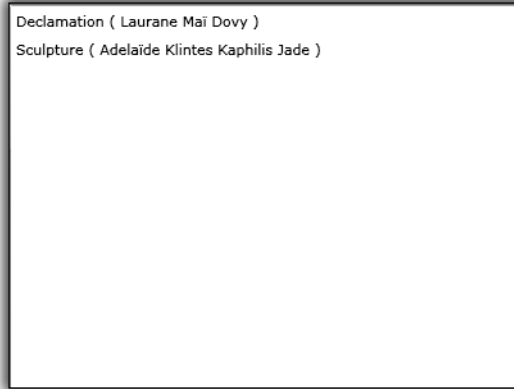
```
public override string ToString()
{
    string ret = this.nom;

    if(etudiants == null)
        return ret;

    ret += "\t(";
    foreach(Etudiant e in etudiants)
        ret += " "+e.Prenom;
    ret += " )";

    return ret;
}
```

Grâce à cette simple modification, notre affichage de données est déjà bien plus agréable à voir :



► Fig. 1.34 :
DataBinding avec
surcharge de ToString()

DataTemplates

Voici la façon la plus intéressante de rendre lisible toute notre liste de cours ainsi que ses élèves respectifs. Un *DataTemplate* est une structure XAML qui sera appliquée à chacun des items d'une *ListBox* donnée.

Ici le changement se fait uniquement dans le code XAML, l'indépendance avec le code applicatif est parfaite. Ce qui permet de déléguer plus facilement ce travail aux designers et intégrateurs.

La *ListBox* possède un attribut du nom de *ItemTemplate* qui permet de définir quel *Template* utiliser pour afficher les items :

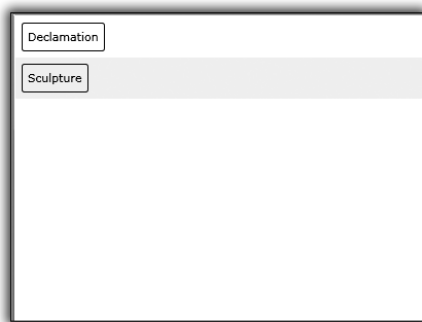
⚙ Exemple de DataTemplate 1

```
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <ListBox Name="CoursListBox">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <Border CornerRadius="2" Margin="3"
                        BorderBrush="Black" BorderThickness="1">
                        <TextBlock Text="{Binding Path=Nom}"
                            Margin="5"/>
                    </Border>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</UserControl>
```

```

        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
</Grid>
</UserControl>

```



► Fig. 1.35 :
Exemple de
DataTemplate 1

Pour préciser quelles propriétés des objets forment la source du binding, il faut afficher. XAML offre une méthode de navigation par chaîne de caractères à travers ces derniers.

Ainsi le code `{Binding Path=Nom}` dans notre exemple, stipule que pour chaque *Cours*, l'interface doit afficher la propriété *Nom*.

En ce qui concerne la liste d'élèves, il faut ajouter à ce *DataTemplate* une nouvelle liste, dont l'*ItemSource* est la propriété *Etudiants* de chaque cours :

⚙ Exemple de DataTemplate2

```

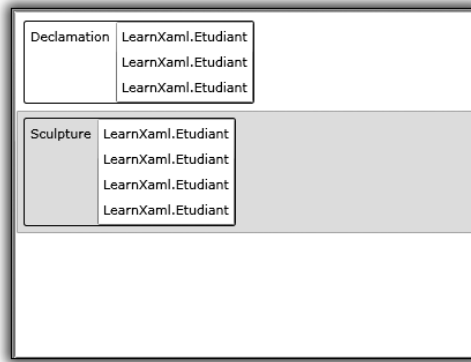
<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <ListBox Name="CoursListBox">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <Border CornerRadius="2" Margin="3"
                        BorderBrush="Black" BorderThickness="1">
                        <StackPanel Orientation="Horizontal">
                            <TextBlock Text="{Binding Path=Nom}" Margin="5"/>
                            <ListBox ItemsSource="{Binding Path=Etudiants}"/>
                        </StackPanel>
                    </Border>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</UserControl>

```

```

    </ListBox>
  </Grid>
</UserControl>

```



► Fig. 1.36 :
Exemple de
DataTemplate2

Un nouveau *DataTemplate* doit ensuite être créé, cette fois non pour les cours mais pour les étudiants :

⚙️ Exemple de DataTemplate 3

```

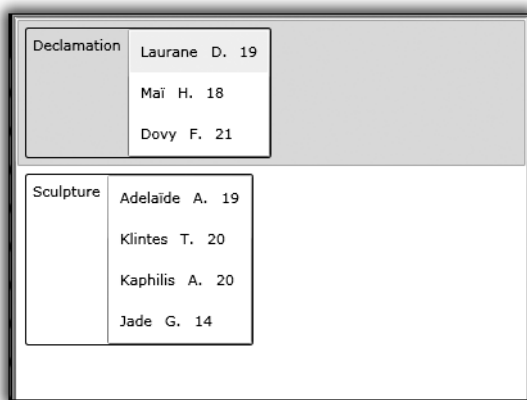
<UserControl x:Class="LearnXAML.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
    <ListBox Name="CoursListBox">
      <ListBox.ItemTemplate>
        <DataTemplate>
          <Border CornerRadius="2" Margin="3"
            BorderBrush="Black" BorderThickness="1">
            <StackPanel Orientation="Horizontal">
              <TextBlock Text="{Binding Path=Nom}" Margin="5"/>
              <ListBox ItemsSource="{Binding Path=Etudiants}">
                <ListBox.ItemTemplate>
                  <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                      <TextBlock Text="{Binding
                        Path=Prenom}" Margin="5"/>
                      <TextBlock Text="{Binding
                        Path=Nom}" Margin="5"/>
                      <TextBlock Text="{Binding
                        Path=Age}" Margin="5"/>
                    </StackPanel>
                  </DataTemplate>
                </ListBox.ItemTemplate>
              </ListBox>
            </StackPanel>
          </Border>
        </DataTemplate>
      </ListBox.ItemTemplate>
    </ListBox>
  </Grid>
</UserControl>

```

```

        </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</StackPanel>
</Border>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</UserControl>

```



► Fig. 1.37 :
Exemple de
DataTemplate 3

Cette méthode donne un résultat surprenant. Cependant, le code, lui, est des plus immenses. Lire un code XAML de ce genre est presque impossible dès qu'il grandit un peu.

Résoudre ce problème de clarté invoque l'utilisation des *Ressources* XAML.

Chaque élément XAML a la possibilité d'être l'hôte d'une collection de ressources. Ces ressources seront accessibles à tous les éléments enfants de l'élément hôte grâce à leur nom. (En l'occurrence, grâce à leur `x:Key`.)

Le code XAML de l'exemple précédent, agrémenté de l'utilisation des ressources, devient :



Exemple de DataTemplate en Ressources

```

<UserControl x:Class="LearnXAML.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/XAML/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/XAML"
    Width="400" Height="300">

```



```

<UserControl.Resources>

    <DataTemplate x:Key="EtudiantDataTemplate">
        <StackPanel Orientation="Horizontal">
            <TextBlock Text="{Binding Path=Prenom}" Margin="5"/>
            <TextBlock Text="{Binding Path=Nom}" Margin="5"/>
            <TextBlock Text="{Binding Path=Age}" Margin="5"/>
        </StackPanel>
    </DataTemplate>

    <DataTemplate x:Key="CoursDataTemplate">
        <Border CornerRadius="2" Margin="3"
            BorderBrush="Black" BorderThickness="1">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Path=Nom}" Margin="5"/>
                <ListBox ItemsSource="{Binding Path=Etudiants}"
                    ItemTemplate="{StaticResource EtudiantDataTemplate}"/>
            </StackPanel>
        </Border>
    </DataTemplate>

</UserControl.Resources>

<Grid x:Name="LayoutRoot" Background="White">
    <ListBox Name="CoursListBox"
        ItemTemplate="{StaticResource CoursDataTemplate}"/>
</Grid>
</UserControl>

```

Non seulement la lisibilité du code s'est beaucoup améliorée, mais en plus, ces *DataTemplates* sont maintenant réutilisables à d'autres endroits de notre interface, sans qu'on ait besoin de les copier.

Différence entre StaticResource et DynamicResource

Remarquez l'utilisation du mot *StaticResource*. Une ressource statique est une ressource définie dans le même document et en amont de son utilisation. Ceci explique que *EtudiantDataTemplate* soit défini avant *CoursDataTemplate*.

Une alternative à cette structure rigoureuse est d'utiliser *DynamicResource*. Une ressource dynamique peut être définie n'importe où dans l'application, même dans un autre fichier.

ValueConverter

Dans l'exemple précédent, une propriété *C#* est affichée telle quelle lors de son *binding* sur l'interface. Ainsi, le nom *Lauranne* est pleinement lisible.

Cependant ce n'est pas toujours l'idéal. Si nous ajoutons la propriété *EstDoué* de type *booléen* aux étudiants, le résultat de son *binding* sera soit *TRUE*, soit *FALSE* :



Ajout à Etudiant.cs

```
private bool estDoué;

public bool EstDoué
{
    get { return estDoué; }
    set { estDoué = value; }
}
```

Une fois de plus, la méthode *ToString* du type *booléen* sert à la plateforme.

Declamation	Laurane D. 19 True
	Mai H. 18 True
	Dovy F. 21 False
Sculpture	Adelaïde A. 19 True
	Klintes T. 20 False
	Kaphilis A. 20 False
	Jade G. 14 False

► Fig. 1.38 :
Binding sans
ValueConverter

Comme il n'est pas possible de surcharger cette méthode dans un type primitif, une autre solution s'offre à nous : un *ValueConverter*.

Un *ValueConverter* est une classe *C#* qui, comme son nom l'indique, transforme une valeur, quelle qu'elle soit, en une autre valeur.

Ainsi dans le cas présent, nous allons écrire un *ValueConvertteur* transformant la valeur *TRUE* en *est un élève doué*, et la valeur *FALSE* en *n'est pas un élève doué*.

L'interface *IValueConverter* fait d'une classe un *ValueConverteur*. Son implémentation demande deux méthodes :

- Convert convertit la valeur source en valeur à afficher.
- ConvertBack convertit la valeur à afficher en valeur source.

EstDouéValueConverter.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Data;
using System.Globalization;

namespace LearnXaml
{
    public class EstDouéValueConverter : IValueConverter
    {
        private const string Doué = "est un élève doué";
        private const string NonDoué = "n'est pas un élève doué";

        #region IValueConverter Membres

        public object Convert(object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            bool? EstDoué = (value as bool?);

            if (EstDoué == null || EstDoué.HasValue == false)
                throw new InvalidCastException(
                    "EstDouéValueConverter.Convert value is not bool or is null");

            if (EstDoué.Value)
            {
                return Doué;
            }
            else
            {
                return NonDoué;
            }
        }
    }
}
```

```

    }

    }

    public object ConvertBack(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        string EstDoué = value.ToString();
        switch (EstDoué)
        {
            case Doué:
                return true;

            case NonDoué:
                return false;

            default:
                throw new InvalidCastException(
                    "EstDouéValueConverter.ConvertBack value incorrecte" +
                    value.ToString());
        }
    }

    #endregion
}

```

Pour utiliser le *ValueConverter* *EstDouéValueConverter* dans un *binding*, plusieurs étapes doivent être respectées :

- ajouter le code du *ValueConverter* à la solution *Silverlight* (ou en ajouter la référence d'une *assembly* compilée préalablement pour *Silverlight*) ;
- ajouter au fichier XAML un *using* (xmls) pointant vers l'espace de noms du *ValueConverter* (dans ce cas, l'espace de noms est le même que celui de l'application mais cela peut varier) ;
- créer une instance de ce *ValueConverter* dans les ressources de l'application ;
- spécifier au *binding* d'utiliser cette instance de *ValueConverter*.



Modification du fichier XAML

```

<UserControl x:Class="LearnXaml.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:LearnXaml"
    Width="400" Height="300">

```

```

<UserControl.Resources>

    <MyApp:EstDouéValueConverter x:Key="estDouéValueConverter"/>

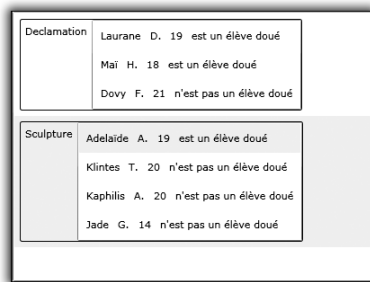
    <DataTemplate x:Key="EtudiantDataTemplate">
        <StackPanel Orientation="Horizontal">
            <TextBlock Text="{Binding Path=Prenom}" Margin="5"/>
            <TextBlock Text="{Binding Path=Nom}" Margin="5"/>
            <TextBlock Text="{Binding Path=Age}" Margin="5"/>
            <TextBlock Text="{Binding Path=EstDoué,
                Converter={StaticResource estDouéValueConverter}}"
                Margin="5" />
        </StackPanel>
    </DataTemplate>

    <DataTemplate x:Key="CoursDataTemplate">
        <Border CornerRadius="2" Margin="3"
            BorderBrush="Black" BorderThickness="1">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Path=Nom}" Margin="5"/>
                <ListBox ItemsSource="{Binding Path=Etudiants}"
                    ItemTemplate="{StaticResource
                        ↳ EtudiantDataTemplate}"/>
            </StackPanel>
        </Border>
    </DataTemplate>

</UserControl.Resources>

<Grid x:Name="LayoutRoot" Background="White">
    <ListBox Name="CoursListBox"
        ItemTemplate="{StaticResource CoursDataTemplate}"/>
</Grid>
</UserControl>

```



► Fig. 1.39 :
Binding avec
ValueConverter

Attributs d'un ValueConverter

Il est parfois utile de laisser au chargé d'interface (celui qui écrit le *XAML*) la possibilité de configurer un *ValueConverter* sans devoir passer par le code *C#*. C'est le cas d'un *RatioValueConverter* multipliant simplement un nombre entier par un autre. Il serait aberrant de devoir créer un *RatioValueConverter* par multiplication souhaitée.

Ajouter un attribut configurable en *XAML* se fait en ajoutant une propriété publique dans la classe du *ValueConverter*, ainsi qu'il est montré dans le code suivant avec la propriété *Ratio* :



RatioConverter.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Data;
using System.Globalization;

namespace LearnXaml
{
    public class RatioValueConverter : IValueConverter
    {
        private int ratio = 5;
        public int Ratio
        {
            get { return ratio; }
            set { ratio = value; }
        }

        public object Convert(object value, Type targetType,
            object parameter, CultureInfo culture)
        {
            return (int)value * ratio;
        }

        public object ConvertBack(object value, Type targetType,
            object parameter, CultureInfo culture)
```

```

    {
        return (int)value / ratio;
    }
}

```

Il est alors possible d'utiliser ce *RatioConverter* dans le code *XAML* pour, par exemple, transformer l'âge (en années) des étudiants en un nombre de mois :



Modification du code XAML

```

<MyApp:EstDouéValueConverter x:Key="estDouéValueConverter"/>
<MyApp:RatioValueConverter x:Key="AnneeEnMoisConverter"
    Ratio="12"/>

<DataTemplate x:Key="EtudiantDataTemplate">
    <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Path=Prenom}" Margin="5"/>
        <TextBlock Text="{Binding Path=Nom}" Margin="5"/>
        <TextBlock Text="{Binding Path=Age,
            Converter={StaticResource AnnéeEnMoisConverter}}"
            Margin="5"/>
        <TextBlock Text="{Binding Path=EstDoué,
            Converter={StaticResource estDouéValueConverter}}"
            Margin="5" />
    </StackPanel>
</DataTemplate>

```

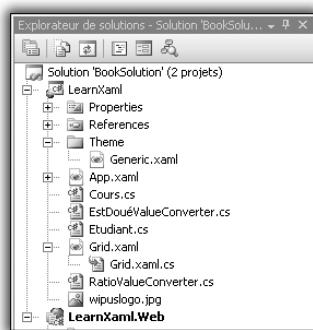
Declamation	Laurane D. 228 est un élève doué
	Maï H. 216 est un élève doué
	Dovy F. 252 n'est pas un élève doué
Sculpture	Adelaïde A. 228 est un élève doué
	Klintes T. 240 n'est pas un élève doué
	Kaphilis A. 240 n'est pas un élève doué
	Jade G. 168 n'est pas un élève doué

► Fig. 1.40 :
Binding avec
RatioValueConverter

Le fichier Generic.XAML

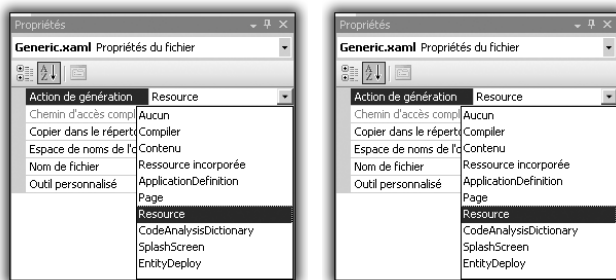
Comme nous le voyons dans l'exemple suivi tout au long ce chapitre, le nombre de ressources de notre page *XAML* ne cesse de grandir.

Pour éviter un code trop long et rendre ces ressources accessibles depuis n'importe quel fichier XAML de notre application, il suffit de créer un fichier *XAML* du nom de *Generic.XAML* dans le dossier *Themes* de l'arborescence de la solution *Silverlight*.



► Fig. 1.41 :
Arborescence de solution pour le fichier
Generic.XAML

Attention, dans les propriétés du fichier *Generic.xaml*, le mode de compilation doit être changé de Page à Resource.



► Fig. 1.42 : Modification des propriétés de Generic.xaml

Redéfinir la structure d'une ListBox

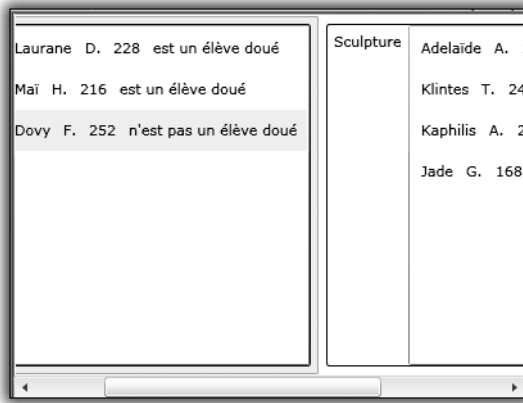
Nous avons vu comment changer l'aspect des items contenus dans une *ListBox* lors d'un *binding*. Il est également possible de forcer l'ordre dans lequel ces items vont s'afficher.

De base, pour une *ListBox*, les items sont affichés dans un *StackPanel* dont l'orientation est verticale. Pour changer cela, il suffit de surcharger l'attribut *ItemPanel* de la *ListBox* :

Surcharge de l'attribut ItemPanel

```
<UserControl x:Class="LearnXaml.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:MyApp="clr-namespace:LearnXaml"
  Width="400" Height="300">
  <UserControl.Resources>
    ...
  </UserControl.Resources>

  <Grid x:Name="LayoutRoot" Background="White">
    <ListBox Name="CoursListBox"
      ItemTemplate="{StaticResource CoursDataTemplate}"
    >
      <ListBox.ItemsPanel>
        <ItemsPanelTemplate>
          <StackPanel Orientation="Horizontal"/>
        </ItemsPanelTemplate>
      </ListBox.ItemsPanel>
    </ListBox>
  </Grid>
</UserControl>
```



► Fig. 1.43 :
Surcharge de l'attribut
ItemPanel

1.9 Colorez votre application grâce aux Brushes et aux Gradients

Ajouter de la couleur à vos applications *Silverlight* se fait grâce aux *Brushes*. Un *Brush* est la représentation d'une couleur.

Nous en avons souvent rencontré dans les exemples précédents de ce livre.

En outre, dans le code suivant, *White* est un *Brush* :



Exemple d'un Brush

```
<UserControl x:Class="LearnXaml.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:MyApp="clr-namespace:LearnXaml"
  Width="400" Height="100">
  <Grid x:Name="LayoutRoot" Background="White">
  </Grid>
</UserControl>
```

De nombreux éléments d'interface ont la possibilité d'être configurés pour utiliser d'autres couleurs que leurs couleurs d'origines.

Il s'agit généralement des attributs *Background*, *Foreground*, *Stroke*, *BorderColor*, etc.

Mais il y a mieux, en plus d'accepter des *Brushes* comme valeurs, ces attributs acceptent aussi des *Gradients*. Les *Gradients* sont des collections de couleurs et d'offsets permettant de créer un dégradé.

La classe *Gradient* est une classe abstraite dont découle quelques types de *Gradients* différents et ne pouvant être utilisés directement en XAML.

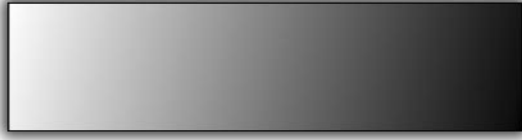
LinearGradientBrush

Le type de *Gradient* le plus simple est le *LinearGradientBrush* :



Exemple de LinearGradientBrush

```
<UserControl x:Class="LearnXaml.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:MyApp="clr-namespace:LearnXaml"
  Width="400" Height="100">
  <Grid x:Name="LayoutRoot">
    <Grid.Background>
      <LinearGradientBrush>
        <GradientStop Offset="0" Color="White"/>
        <GradientStop Offset="1" Color="Black"/>
      </LinearGradientBrush>
    </Grid.Background>
  </Grid>
</UserControl>
```



► Fig. 1.44 :
Exemple de
LinearGradientBrush

Le LinearGradientBrush est une fonction linéaire dont les couleurs varient proportionnellement à un offset. Dans ce cas, le pinceau va varier du blanc à $x = 0$ au noir à $x = 100$.

Il est possible d'ajouter plus de couleurs dans un LinearGradientBrush la fonction est alors divisée en un set de segments de taille suffisante pour afficher le dégradé selon les écrits du XAML :

⚙️ Exemple de LinearGradientBrush découpé

```
<UserControl x:Class="LearnXaml.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:MyApp="clr-namespace:LearnXaml"
  Width="400" Height="100">
  <Grid x:Name="LayoutRoot">
    <Grid.Background>
      <LinearGradientBrush>
        <GradientStop Offset="0" Color="White"/>
        <GradientStop Offset="0.3" Color="Black"/>
        <GradientStop Offset="0.5" Color="White"/>
        <GradientStop Offset="0.7" Color="Black"/>
        <GradientStop Offset="0.9" Color="Black"/>
        <GradientStop Offset="1" Color="White"/>
      </LinearGradientBrush>
    </Grid.Background>
  </Grid>
</UserControl>
```



► Fig. 1.45 :
Exemple de
LinearGradientBrush
découpé

L'orientation du dégradé, ici de gauche à droite, est elle aussi modifiable. Le dégradé commence à son attribut `StartPoint` et finit à son attribut `EndPoint`. Chacun de ces attributs sont des points définis par deux valeurs : X et Y en pourcentage de l'élément à colorier.

En modifiant un peu ces attributs, nous pouvons créer un dégradé vertical :



LinearGradientBrush vertical

```
<UserControl x:Class="LearnXaml.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:MyApp="clr-namespace:LearnXaml"
  Width="400" Height="100">
  <Grid x:Name="LayoutRoot">
    <Grid.Background>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
        <GradientStop Offset="0" Color="White"/>
        <GradientStop Offset="1" Color="Black"/>
      </LinearGradientBrush>
    </Grid.Background>
  </Grid>
</UserControl>
```



► Fig. 1.46 :
LinearGradientBrush
vertical

RadialGradientBrush

Un *RadialGradientBrush* agit de la même manière qu'un *LinearGradientBrush* mais en décrivant des cercles :



Exemple de RadialGradientBrush

```
<UserControl x:Class="LearnXaml.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:MyApp="clr-namespace:LearnXaml"
  Width="400" Height="100">
  <Grid x:Name="LayoutRoot">
    <Grid.Background>
      <RadialGradientBrush>
```

```

        <GradientStop Offset="0" Color="White"/>
        <GradientStop Offset="0.3" Color="Black"/>
        <GradientStop Offset="0.5" Color="White"/>
        <GradientStop Offset="0.7" Color="Black"/>
        <GradientStop Offset="0.9" Color="Black"/>
        <GradientStop Offset="1" Color="White"/>
    </RadialGradientBrush>
</Grid.Background>
</Grid>
</UserControl>

```



► Fig. 1.47 :
Exemple de
RadialGradientBrush

Contrairement au *LinearGradientBrush*, ce sont les attributs *Center* et *GradientOrigin* qui permettent de changer la source et l'orientation du dégradé :

⚙️ RadialGradientBrush décentré

```

<UserControl x:Class="LearnXaml.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:LearnXaml"
    Width="400" Height="100">
    <Grid x:Name="LayoutRoot">
        <Grid.Background>
            <RadialGradientBrush Center="0.8,0.8"
                ➤ GradientOrigin="0.8,0.8">
                <GradientStop Offset="0" Color="White"/>
                <GradientStop Offset="1" Color="Black"/>
            </RadialGradientBrush>
        </Grid.Background>
    </Grid>
</UserControl>

```



► Fig. 1.48 :
RadialGradientBrush
décentré

ImageBrush

Les *ImageBrushes* vous permettent d'utiliser une image à la place d'un *Brush*. On les emploie de la même façon que l'élément *Image* :

Exemple de ImageBrush

```
<UserControl x:Class="LearnXaml.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:LearnXaml"
    Width="400" Height="100">
    <Grid x:Name="LayoutRoot">
        <Grid.Background>
            <ImageBrush ImageSource="wipuslogo.jpg" Stretch="None"/>
        </Grid.Background>
    </Grid>
</UserControl>
```



► Fig. 1.49 :
Exemple d'ImageBrush

L'utilité de ce genre de *GradientBrush* est évidente lorsqu'on l'emploie sur des attributs plus surprenants de la plateforme *Silverlight*. Dans l'exemple qui suit, le même *ImageBrush* est utilisé sur l'attribut *Foreground* d'une *TextBox* :

Exemple d'ImageBrush sur l'attribut Foreground

```
<UserControl x:Class="LearnXaml.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:LearnXaml"
    Width="400" Height="100">
    <Grid x:Name="LayoutRoot" Background="Black">
        <TextBlock Text="WIPUS" FontSize="100"
            HorizontalAlignment="Center"
            VerticalAlignment="Center">
            <TextBlock.Foreground>
                <ImageBrush ImageSource="wipuslogo.jpg"
                    Stretch="UniformToFill"/>
            </TextBlock.Foreground>
        </TextBlock>
    </Grid>
</UserControl>
```

```
</TextBlock>
</Grid>
</UserControl>
```



► Fig. 1.50 :
Exemple d'ImageBrush
sur l'attribut Foreground

1.10 Animez votre application grâce aux StoryBoard

En XAML, il est possible d'animer n'importe quoi, n'importe comment. En effet, par animation, le XAML n'entend pas seulement mouvement d'un objet mais bien variation d'une valeur à partir d'une valeur initiale jusqu'à une valeur finale sur une période de temps donnée.

Ainsi, bien qu'il soit possible de faire bouger un élément grâce à une animation, il est également envisageable d'en faire varier sa couleur, sa taille, etc.

Plusieurs animations peuvent être réunies en un ensemble appelé *Storyboard*, scénario en français.

Pour commencer en beauté, voici un scénario qui va changer la taille du texte d'un *TextBlock* et déplacer la position d'un rectangle :

Les animations Silverlight (XAML)

```
<UserControl x:Class="LearnXaml.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:MyApp="clr-namespace:LearnXaml"
  Width="400" Height="100">
  <Grid x:Name="LayoutRoot" Background="Black">
    <TextBlock Name="HeodeLabel"
      MouseLeftButtonDown="HeodeLabel_Click"
      Text="Heode" FontSize="10"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Foreground="White">
    </TextBlock>
    <Canvas>
      <Rectangle Fill="LimeGreen"
        Name="ProgressRect"
        Height="20">
```

```

        Width="40"
        Canvas.Top="80"
        Canvas.Left="0"
        RadiusX="10"
        RadiusY="10"/>
</Canvas>
<Grid.Resources>
    <Storyboard x:Name="MagnifyStoryboard"
        Duration="00:00:10"
        RepeatBehavior="Forever"
        AutoReverse="True">
        <DoubleAnimationUsingKeyFrames
            Storyboard.TargetName="HeodeLabel"
            Storyboard.TargetProperty="FontSize"
            BeginTime="0" Duration="5">
            <SplineDoubleKeyFrame KeyTime="00:00:00"
                Value="10"/>
            <SplineDoubleKeyFrame KeyTime="00:00:10"
                Value="100"/>
        </DoubleAnimationUsingKeyFrames>
        <DoubleAnimationUsingKeyFrames
            Storyboard.TargetName="ProgressRect"
            Storyboard.TargetProperty="(Canvas.Left)">
            <SplineDoubleKeyFrame KeyTime="00:00:00"
                Value="0"/>
            <SplineDoubleKeyFrame KeyTime="00:00:10"
                Value="360"/>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>
</Grid.Resources>
</Grid>
</UserControl>

```

Cette animation est déclenchée lors de l'événement `MouseLeftButtonClic` sur le *TextBox* *HeodeLabel*.

Le code de la logique applicative de cette application appelle la méthode `Begin` du scénario *MagnifyStoryboard*. Cette méthode va enclencher simultanément chacune des animations contenues dans ce scénario :

⚙️ Les animation en Silverlight (C#)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace LearnXaml
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }

        private void HeodeLabel_Clic(object sender, MouseButtonEventArgs e)
        {
            MagnifyStoryboard.Begin();
        }
    }
}
```



► Fig. 1.51 :
Animation non démarrée



► Fig. 1.52 :
Animation après
quelques secondes



► Fig. 1.53 :
Animation après
10 secondes

Ce scénario a une durée (*Duration*), une règle de répétition (*RepeatBehavior*) et une règle de renversement (*AutoReverse*).

La durée est le temps que prend un cycle complet de ce scénario ; il s'agit d'une valeur temporelle représentée sous la forme hh:mm:ss.

La règle de répétition est elle aussi une valeur temporelle. Si cette règle avait eu comme valeur 10:00:00, ce scénario se répéterait pendant une période de 10 heures suivant l'appel de la méthode *Begin*, et s'arrêterait ensuite.

Dans le cas présent, la valeur de la règle de répétition est *Forever* ; cela signifie que le scénario redémarrera *ad vitam aeternam*.

La règle de renversement, quant à elle, est une valeur booléenne. Lorsqu'elle est vraie, une fois arrivée à sa fin, le scénario sera rembobiné.

Une *DoubleAnimationUsingKeyFrame* est une animation modifiant la valeur d'un double (nombre réel) sur une certaine durée.

Ses attributs *Storyboard.TargetName* et *Storyboard.TargetProperty* indiquent respectivement quel élément *XAML* contient ce double, et le nom de l'attribut représentant ce double.

Ces animations possèdent un attribut *BeginTime* et une durée (*Duration*).

Inutile de préciser que jouer avec ces *Storyboard* est une bonne solution pour faire luire un bouton lorsque la souris passe par-dessus, et ainsi de suite.

Créez une bannière Silverlight grâce aux animations

Grâce au savoir acquis au cours de ce chapitre, il nous est maintenant possible de créer une bannière *Silverlight* tel que celle du site de Microsoft.



► Fig. 1.54 : Bannière du site Microsoft

Cette bannière affiche 3 publicités différentes ; des boutons sur le côté permettent de changer de publicité. Lorsque la souris passe par-dessus un de ces boutons, les publicités glissent les unes sur les autres jusqu'à ce que la publicité relative au bouton survolé soit visible.

Commençons par définir l'interface visuelle de la bannière sous la structure suivante :

- Canvas
 - StackPanel PubCanvas1
 - .Publicité1 (Image + Lien vers site web)
 - .Bouton vers Publicité1
 - StackPanel PubCanvas2
 - .Publicité2 (Image + Lien vers site web)
 - .Bouton vers Publicité2
 - StackPanel PubCanvas3
 - .Publicité3 (Image + Lien vers site web)
 - .Bouton vers Publicité3

⚙ Interface visuelle de la bannière

```
<UserControl x:Class="SilverlightBanner.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="500" Height="200">
    <Canvas x:Name="LayoutRoot" Background="White">

        <StackPanel Name="PubCanvas1" Orientation="Horizontal">
            <Border Name="ToPub1" Background="Beige"
                BorderBrush="Black"
                BorderThickness="2"
                CornerRadius="3"
                Height="200"
                Width="80"
                MouseMove="ToPub1_MouseEnter"
                MouseEnter="ToPub1_MouseEnter">
                <Canvas>

                    <TextBlock Text="Wipus"
                        Canvas.Top="80"
                        Canvas.Left="50"
                        FontSize="16">

                        <TextBlock.RenderTransform>
                            <RotateTransform Angle="90"
                                CenterX="0"
                                CenterY="0"/>
                        </TextBlock.RenderTransform>
                    </TextBlock>
                </Canvas>
            </Border>
            <Border Name="Publicité1"
                Background="White"
                Height="200"
                Width="260"
                Opacity="1">
                <Canvas>
                    <Image Source="wipuslogo.jpg" Stretch="Uniform"
                        Height="200"
                        Width="160"/>
                    <TextBlock Text="Wipus" FontSize="35"
                        Canvas.Top="100"
                        FontFamily="Arial"
                        Canvas.Left="100"/>
                    <HyperlinkButton NavigateUri="http:\\www.wipus.com"
                        Canvas.Left="110" Canvas.Top="130">
                </StackPanel Orientation="Horizontal">
```

```

        <Image Source="smallbtt.jpg"/>
        <TextBlock Text="Tell me more..."
            Foreground="Black"
            Margin="5"
            FontSize="14"/>
    </StackPanel>
</HyperlinkButton>
</Canvas>
</Border>
</StackPanel>

<StackPanel Name="PubCanvas2" Orientation="Horizontal"
    Canvas.Left="80">
    <Border Name="ToPub2" Background="Beige"
        BorderBrush="Black"
        BorderThickness="2"
        CornerRadius="3"
        Height="200"
        Width="80"
        MouseMove="ToPub2_MouseEnter"
        MouseEnter="ToPub2_MouseEnter">
        <Canvas>
            <TextBlock Text="Heode"
                Canvas.Top="80"
                Canvas.Left="50"
                FontSize="16">
            <TextBlock.RenderTransform>
                <RotateTransform Angle="90"
                    CenterX="0"
                    CenterY="0"/>
            </TextBlock.RenderTransform>
            </TextBlock>
        </Canvas>
    </Border>
    <Border Name="Publicité2"
        Background="White"
        Height="200"
        Width="260"
        Opacity="1">
        <Canvas>
            <Image Source="heodelogo.jpg" Stretch="Uniform"
                Height="200"
                Width="160"/>
            <TextBlock Text="Heode" FontSize="35"
                Canvas.Top="100"
                FontFamily="Arial"
                Canvas.Left="100"/>
        </Canvas>
    </Border>
</StackPanel>

```

```

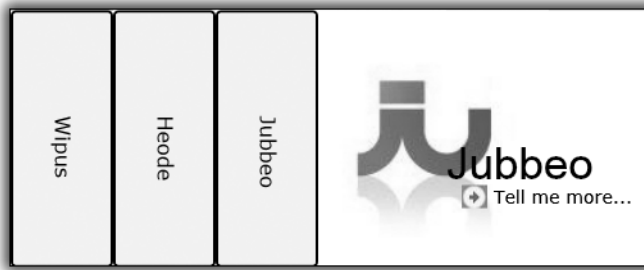
        <HyperlinkButton NavigateUri="http:\\www.heode.com"
            Canvas.Left="110" Canvas.Top="130">
            <StackPanel Orientation="Horizontal">
                <Image Source="smallbtt.jpg"/>
                <TextBlock Text="Tell me more..."
                    Foreground="Black"
                    Margin="5"
                    FontSize="14"/>
            </StackPanel>
        </HyperlinkButton>
    </Canvas>
</Border>
</StackPanel>

<StackPanel Name="PubCanvas3" Orientation="Horizontal"
    Canvas.Left="160">
    <Border Name="ToPub3" Background="Beige"
        BorderBrush="Black"
        BorderThickness="2"
        CornerRadius="3"
        Height="200"
        Width="80"
        MouseMove="ToPub3_MouseEnter"
        MouseEnter="ToPub3_MouseEnter"
        Opacity="1">
        <Canvas>
            <TextBlock Text="Jubbeo"
                Canvas.Top="80"
                Canvas.Left="50"
                FontSize="16">
            <TextBlock.RenderTransform>
                <RotateTransform Angle="90"
                    CenterX="0"
                    CenterY="0"/>
            </TextBlock.RenderTransform>
        </TextBlock>
    </Canvas>
    </Border>
    <Border Name="Publicité3"
        Background="White"
        Height="200"
        Width="260"
        Opacity="1">
        <Canvas>
            <Image Source="jubbeologo.jpg" Stretch="Uniform"
                Height="200"
                Width="160"/>

```

```
<TextBlock Text="Jubbeo" FontSize="35"
    Canvas.Top="100"
    FontFamily="Arial"
    Canvas.Left="100"/>
<HyperlinkButton NavigateUri="http:\\www.jubbeo.com"
    Canvas.Left="110" Canvas.Top="130">
    <StackPanel Orientation="Horizontal">
        <Image Source="smallbtt.jpg"/>
        <TextBlock Text="Tell me more..."
            Foreground="Black"
            Margin="5"
            FontSize="14"/>
    </StackPanel>
</HyperlinkButton>
</Canvas>
</Border>
</StackPanel>

</Canvas>
</UserControl>
```



► Fig. 1.55 :
Interface visuelle de la
bannière

L'effet voulu est le suivant : à chaque passage sur un des Border ToPub1, ToPub2 ou ToPub3, les publicités vont glisser vers la droite pour afficher la publicité relative au Border survolé.

Si la publicité relative au Border a déjà glissé préalablement vers la droite, elle va glisser vers la gauche.

Il y a donc 3 états à notre bannière :

- Publicité3Visible (état actuel de la bannière) ;
- Publicité2Visible ;
- Publicité1Visible.

Pour des raisons de lisibilité du code, créons une énumération représentant ces états :

Etat de la bannière (C#)

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SilverlightBanner
{
    public enum EtatBannière
    {
        Publicité1Visible,
        Publicité2Visible,
        Publicité3Visible
    }
}
```

Ce sont des animations qui vont nous permettre de passer d'un état à l'autre. Pour naviguer entre 3 états, il faut 4 animations (ou scénario en l'occurrence, chaque scénario contenant une unique animation).

Ces animations sont :

- DePub3àPub2 ;
- DePub2àPub1 ;
- DePub1àPub2 ;
- DePub2àPub3.

Ajoutons le code XAML de ces scénarios au fichier *XAML* de l'interface :

Les animations de la bannière

```
<UserControl x:Class="SilverlightBanner.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="500" Height="200">
```



```

<Canvas x:Name="LayoutRoot" Background="White">

<!--( définition de l'interface ...)-->

<Canvas.Resources>

    <Storyboard x:Name="DePub3àPub2">
        <DoubleAnimationUsingKeyFrames
            BeginTime="00:00:00"
            Storyboard.TargetName="PubCanvas3"
            Storyboard.TargetProperty="(Canvas.Left)">
            <SplineDoubleKeyFrame KeyTime="00:00:00" Value="160"/>
            <SplineDoubleKeyFrame KeyTime="00:00:04" Value="420"/>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>

    <Storyboard x:Name="DePub2àPub1">
        <DoubleAnimationUsingKeyFrames
            BeginTime="00:00:00"
            Storyboard.TargetName="PubCanvas2"
            Storyboard.TargetProperty="(Canvas.Left)">
            <SplineDoubleKeyFrame KeyTime="00:00:00" Value="80"/>
            <SplineDoubleKeyFrame KeyTime="00:00:04" Value="340"/>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>

    <Storyboard x:Name="DePub1àPub2">
        <DoubleAnimationUsingKeyFrames
            BeginTime="00:00:00"
            Storyboard.TargetName="PubCanvas2"
            Storyboard.TargetProperty="(Canvas.Left)">
            <SplineDoubleKeyFrame KeyTime="00:00:00" Value="340"/>
            <SplineDoubleKeyFrame KeyTime="00:00:04" Value="80"/>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>

    <Storyboard x:Name="DePub2àPub3">
        <DoubleAnimationUsingKeyFrames
            BeginTime="00:00:00"
            Storyboard.TargetName="PubCanvas3"
            Storyboard.TargetProperty="(Canvas.Left)">
            <SplineDoubleKeyFrame KeyTime="00:00:00" Value="420"/>
            <SplineDoubleKeyFrame KeyTime="00:00:04" Value="160"/>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>
</Canvas.Resources>

```

```
</Canvas>
</UserControl>
```

C'est ensuite sur les événements `MouseMove` et `MouseEnter` de chaque *Border ToPub* qu'il faut ajouter le déclenchement d'un de ces scénarios.

Une analyse préalable de la logique applicative semble requise :



Analyse de la logique applicative de la bannière

Déclarer un **état** assigné à **PublicitéVisible** au démarrage de l'application

Déclarer un **booléen** signalant si une animation est déjà en cours ou non pour éviter que 2 **animations** ne démarrent en même temps

```
{
    Ajouter à chaque fin d'animation un évènement modifiant
    ce booléen

    Modifier ce booléen avant chaque appel de la méthode Begin
    d'un animation
}
```

Déclarer une **destination** de type **EtatBannière**.

Cette destination est la dernière publicité survolée par la souris de l'utilisateur.

Créer une méthode **StartNextAnimation** qui démarre une **animation** en fonction des valeurs **état** et **destination**.

Appeler cette méthode à chaque fin d'animation au cas où l'**état** soit actuel différent de la **destination** actuelle

Lors du survol des Bords **ToPub**.

```
{
    Assigner la publicité courante comme destination
    Si aucune animation n'est en cours, alors appeler la méthode
    StartNextAnimation
}
```

Code de la logique applicative :

Code C# de la bannière

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SilverlightBanner
{
    public partial class Page : UserControl
    {
        private EtatBannière état = EtatBannière.Publicité3Visible;
        private bool AnimationEnCours = false;
        private EtatBannière destination =
            ➤ EtatBannière.Publicité3Visible;

        public Page()
        {
            InitializeComponent();

            DePub1àPub2.Completed += new EventHandler(SetEtatToPub2);
            DePub2àPub1.Completed += new EventHandler(SetEtatToPub1);
            DePub2àPub3.Completed += new EventHandler(SetEtatToPub3);
            DePub3àPub2.Completed += new EventHandler(SetEtatToPub2);
        }

        void SetEtatToPub1(object sender, EventArgs e)
        {
            état = EtatBannière.Publicité1Visible;
            AnimationEnCours = false;
            StartNextAnimation();
        }

        void SetEtatToPub2(object sender, EventArgs e)
        {
            état = EtatBannière.Publicité2Visible;
            AnimationEnCours = false;
            StartNextAnimation();
        }
    }
}
```

```

    }

    void SetEtatToPub3(object sender, EventArgs e)
    {
        état = EtatBannière.Publicité3Visible;
        AnimationEnCours = false;
        StartNextAnimation();
    }

    private void ToPub3_MouseEnter(object sender, MouseEventArgs e)
    {
        destination = EtatBannière.Publicité3Visible;

        if (AnimationEnCours == true)
            return;

        StartNextAnimation();
    }

    private void ToPub2_MouseEnter(object sender, MouseEventArgs e)
    {
        destination = EtatBannière.Publicité2Visible;

        if (AnimationEnCours == true)
            return;

        StartNextAnimation();
    }

    private void ToPub1_MouseEnter(object sender, MouseEventArgs e)
    {
        destination = EtatBannière.Publicité1Visible;

        if (AnimationEnCours == true)
            return;

        StartNextAnimation();
    }

    private void StartNextAnimation()
    {
        if (état == destination)
            return;

        AnimationEnCours = true;

        switch (état)

```

```

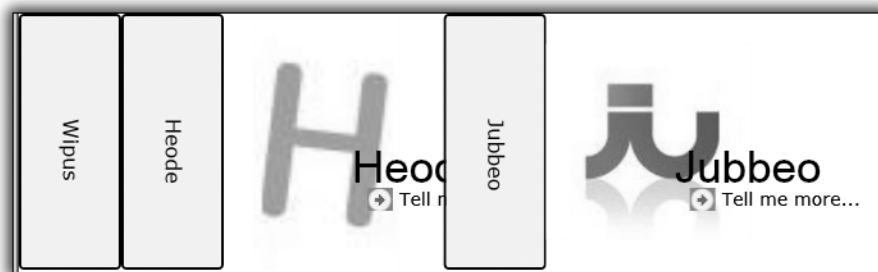
    {
        case EtatBannière.Publicité1Visible:
            DePub1àPub2.Begin();
            break;

        case EtatBannière.Publicité2Visible:
            if (destination == EtatBannière.Publicité1Visible)
                DePub2àPub1.Begin();
            else
                DePub2àPub3.Begin();
            break;

        case EtatBannière.Publicité3Visible:
            DePub3àPub2.Begin();
            break;
    }
}
    }
}
    }
}

```

La bannière devrait maintenant fonctionner à merveille ; cependant, ce n'est pas le cas. Lors de l'exécution du premier scénario, la publicité 3 sort du contrôle *Silverlight* et empiète sur la page *ASP.NET* (ou *HTML*) le contenant.



► Fig. 1.56 : Animation sortant du contrôle Silverlight

Ce problème n'est pas un problème dû au code *Silverlight* mais bien un problème dû au code *ASP.NET* (ou *HTML*) généré par *VisualStudio*.

Dans la page *ASP.NET* hôte du projet de développement *Silverlight*, remplacez la taille du contrôle *Silverlight* par `With =500` et `Height=200` :

⚙ Page ASP.NET hôte d'un projet de développement Silverlight

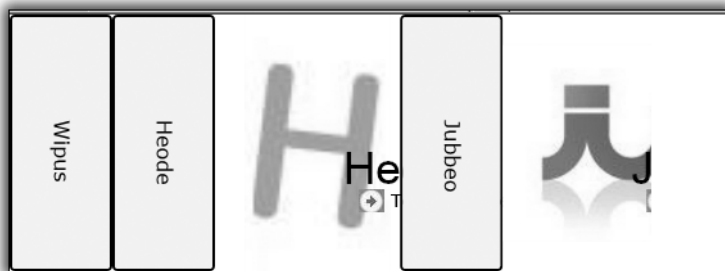
```
<%@ Page Language="C#" AutoEventWireup="true" %>

<%@ Register Assembly="System.Web.Silverlight"
    Namespace="System.Web.UI.SilverlightControls"
    TagPrefix="asp" %>

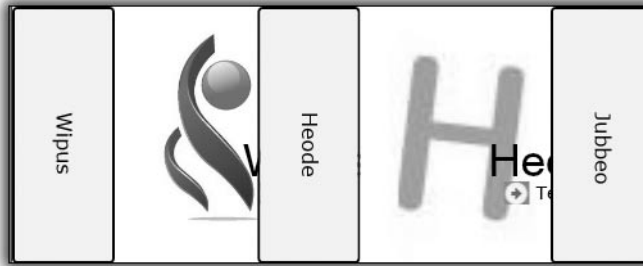
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%;">
<head runat="server">
    <title>SilverlightBanner</title>
</head>
<body style="height:100%;margin:0;">
    <form id="form1" runat="server" style="height:100%;">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <div style="height:100%">
            <asp:Silverlight ID="Xaml1" runat="server"
                Source="~/ClientBin/SilverlightBanner.xap"
                MinimumVersion="2.0.31005.0"
                Width="100%" Height="100%" />
        </div>
    </form>
</body>
</html>
```

La bannière fonctionnera parfaitement.



► Fig. 1.57 : Bannière état 3 à 2



► Fig. 1.58 :
Bannière état 2 à 1



► Fig. 1.59 :
Bannière état 1

1.11 Check-List

Ce chapitre vous a fourni les bases nécessaires à la compréhension d'un fichier *XAML*. Nous avons étudié ici les différents contrôles et fonctionnalités de base d'une interface *Silverlight* :

- ✓ le langage *XAML* dérivé du *XML* ;
- ✓ les différents contrôles de *Layout* structurants une interface ;
- ✓ quelques contrôles d'affichages d'informations ;
- ✓ quelques contrôles de captures d'informations ;
- ✓ les bases du *Binding* ;
- ✓ quelques éléments de design.

Enfin, nous avons conclu par la création complète d'une bannière *Silverlight*.

2



2.1 Introduction à Expression Studio	98
2.2 Expression Design	99
2.3 Expression Encoder 2	102
2.4 Expression Blend 2	104
2.5 Intéraction entre Expression Blend et Visual Studio 2008	109
2.6 Check-List	111

Créer vos applications avec Expression Studio

Dans ce chapitre, vous découvrirez la nouvelle gamme d'outils Microsoft destinée aux graphistes et créatifs. Microsoft Expression Studio vous simplifiera la vie lorsque vous devrez créer des applications poussées, au niveau graphique. Découvrez l'ensemble de la gamme d'outils dans ce chapitre : Expression Design pour la partie purement graphique, Expression Web pour tout ce qui concerne l'intégration de Silverlight et ASP.NET dans vos pages, Expression Blend pour la partie animation et web application, Expression Media et Media Encoder pour l'intégration de vos vidéos dans Silverlight.

2.1 Introduction à Expression Studio

Lorsque vous créez une application, que ce soit pour le Web ou pour toute autre plateforme, vous devez veiller à ce que tout soit correctement présenté. Le design, c'est-à-dire l'interface du produit, influence beaucoup les consommateurs. C'est ainsi qu'Apple a, depuis un moment, un succès de plus en plus remarqué grâce au design raffiné de ses ordinateurs.

Jusqu'alors, l'univers des créatifs et des développeurs était différent. D'un côté, les graphistes utilisaient des suites logicielles d'Adobe avec le célèbre Photoshop ou Illustrator et d'un autre côté, les développeurs employaient des logiciels de développement comme Visual Studio que vous avez découvert lors du premier chapitre. Ces logiciels n'avaient aucune cohérence entre eux, ce qui posait énormément de problèmes de communication entre créatifs et développeurs.

Microsoft, soucieux des méthodologies de travail, a donc créé une nouvelle suite de logiciels appelée Expression Studio. Cette suite est plutôt destinée aux créatifs et intégrateurs. Microsoft a réussi son pari pour ces nouveaux logiciels ; non seulement l'interopérabilité avec Visual Studio est parfaite, mais en plus, les créatifs qui ont déjà eu l'occasion de l'utiliser ne sont pas déçus. Seul bémol pour le moment : Expression Studio tourne uniquement sur du Windows. Seul Microsoft Expression Media 2 peut fonctionner dans un environnement Mac OS.

Nul doute que tout le monde n'abandonnera pas la suite Adobe mais vous trouverez sur Internet une série d'outils vous permettant de faire la liaison entre la gamme de logiciels Adobe et la gamme Expression Studio.

La suite Microsoft a pour vocation de faciliter la création de *RIA (Rich Internet Application)*.

Rich Internet Application

Ce terme est apparu avec la redécouverte d'AJAX. Les applications sont dites riches quand elles permettent beaucoup d'interaction avec l'utilisateur. Flash et Silverlight sont orientés vers ce genre d'interactions. En effet, comme tout est téléchargé du côté du client, on peut avoir des applications web très rapides à l'exécution, ce qui permet une très bonne réactivité et donc une bonne interaction avec l'utilisateur de l'application. Le mot *riche* peut aussi être associé à des interfaces jolies ou encore *user friendly*. Nous verrons dans les prochains sous-chapitres que Expression Studio a tout pour créer ce genre d'application.

La gamme Expression Studio comprend 5 outils :

- Expression Web ;
- Expression Blend ;
- Expression Design ;
- Expression Media ;
- Expression Encoder 2.

Les plus utiles, dans le cas de Silverlight, sont Expression Blend, Expression Media et Encoder (les deux derniers sont fortement liés). Si vous ne comptez pas tirer parti de la puissance de Silverlight pour la vidéo, Expression Blend vous suffira.

Expression Studio est payant mais vous disposez d'une version d'évaluation vous permettant de tester le produit. Vous pourrez trouver ce produit en magasin, à la FNAC par exemple ou chez des partenaires Microsoft.

Un bon exemple d'application réalisé en XAML

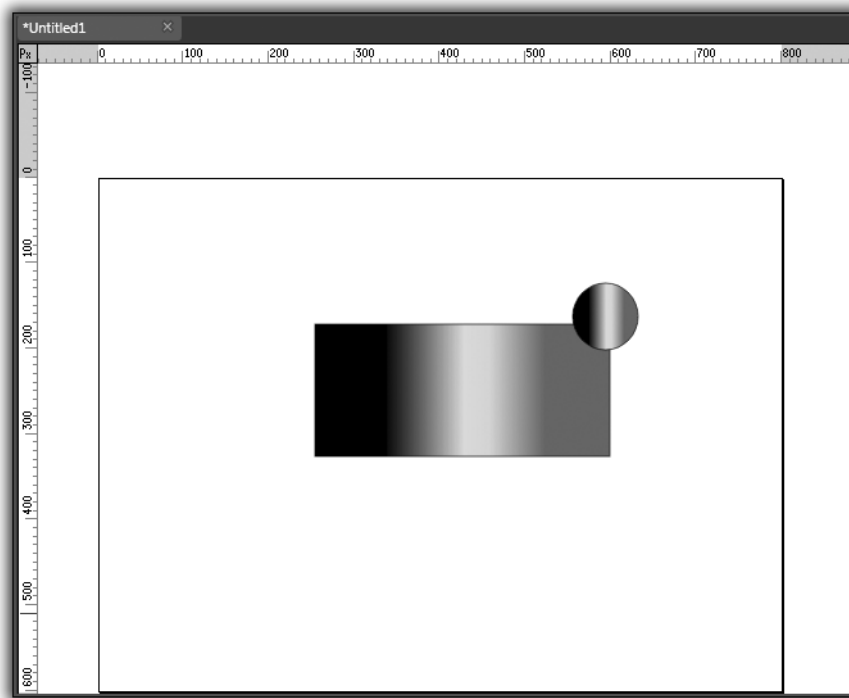
Un bon exemple d'application réalisée en XAML est Expression Studio. En effet, l'ensemble des outils de la gamme Expression a été réalisé en XAML.

Partons maintenant à la découverte de ces 5 outils qui vous serviront tout au long de la création d'une application web Silverlight.

2.2 Expression Design

Expression Design est un peu le Photoshop de Microsoft. Bien que plus jeune et moins complet, il vous permet de réaliser bon nombre d'éléments graphiques. Son grand intérêt par rapport à d'autres logiciels, est sa connaissance par défaut du XAML. Comme vous l'avez vu, XAML est un langage de définitions d'interfaces vectorielles. Expression Design gère uniquement des images vectorielles par défaut, ce qui permet de les convertir très rapidement en XAML.

Prenons un exemple, si dans Expression Design, vous créez un dessin ainsi :



► Fig. 2.1 : Expression Design 2

Si vous voulez créer le même dessin en XAML, voici à quoi cela ressemblera :

```
<Canvas
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Name="Layer_1_0" Width="800" Height="600" Canvas.Left="0"
  Canvas.Top="0">
  <Rectangle
    Width="345.983"
    Height="155.492"
    Canvas.Left="253.257"
    Canvas.Top="170.506"
    Stretch="Fill"
    StrokeLineJoin="Round"
    Stroke="#FF000000">
    <Rectangle.Fill>
      <LinearGradientBrush StartPoint="-0.00144939,0.5"
        EndPoint="1.00145,0.5">
        <LinearGradientBrush.GradientStops>
          <GradientStop Color="#FF000000" Offset="0.237443"/>
        </GradientStops>
      </LinearGradientBrush>
    </Rectangle.Fill>
  </Rectangle>
</Canvas>
```

```

        <GradientStop Color="#FFF3F025" Offset="0.502283"/>
        <GradientStop Color="#FFE9F100" Offset="0.589041"/>
        <GradientStop Color="#FFED2E16" Offset="0.776256"/>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<Ellipse
    Width="77.4962"
    Height="78.9961"
    Canvas.Left="554.742"
    Canvas.Top="122.509"
    Stretch="Fill"
    StrokeLineJoin="Round"
    Stroke="#FF000000">
    <Ellipse.Fill>
        <LinearGradientBrush StartPoint="-0.00653628,0.5"
            EndPoint="1.00654,0.5">
            <LinearGradientBrush.GradientStops>
                <GradientStop Color="#FF000000" Offset="0.237443"/>
                <GradientStop Color="#FFF3F025" Offset="0.502283"/>
                <GradientStop Color="#FFE9F100" Offset="0.589041"/>
                <GradientStop Color="#FFED2E16" Offset="0.776256"/>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Ellipse.Fill>
</Ellipse>
</Canvas>

```

Imaginez alors combien de temps cela vous prendrait de créer des dessins beaucoup plus complexes ? Expression Design est donc un outil indispensable à la réalisation d'interfaces riches.

Expression Design fait l'objet de livres entiers. Si vous êtes habitué à Photoshop, vous n'aurez aucun problème à le prendre en main.

XAML vs Images

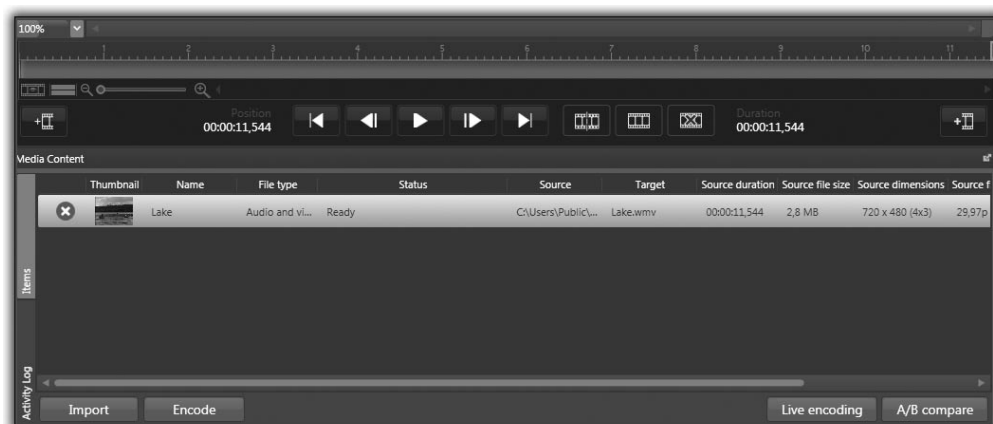
Si XAML représente une image, pourquoi ne pas directement prendre cette image et l'ajouter comme ressource à notre projet en utilisant la balise XAML **Image** ? Tout dépend du type d'applications que vous voulez réaliser. Si l'image fait partie intégrante de votre interface et que celle-ci soit extensible, vous devez la prévoir en XAML. Mais si vous devez afficher une série d'articles avec des images, il est inutile de transformer ces images en XAML.

2.3 Expression Encoder 2

Si vous aimez la vidéo et que vous vouliez utiliser vos vidéos dans Silverlight, Expression Encoder 2 est l'outil dont vous avez besoin.

Un peu complexe à prendre en main au début, il vous permet d'effectuer tout le processus du début à la fin, c'est-à-dire de l'encodage de la vidéo à un rendu dans un lecteur Silverlight prêt à l'emploi.

Prenons un exemple avec une vidéo WMV. Vous pouvez l'importer dans Expression Encoder à l'aide du bouton **import**.



► Fig. 2.2 : Expression Encoder

Une fois la vidéo importée, vous pouvez la visualiser à l'aide du lecteur intégré au programme. Vous avez la possibilité de couper la vidéo comme vous le désirez pour y placer des chapitres ou des explications.

Lorsque votre édition est terminée, vous pouvez regarder toute une série de paramètres à droite. Dans le dernier onglet, **output**, vous avez la possibilité de choisir un lecteur pour votre vidéo. Il en existe toute une série et vous pouvez en créer vous-même de nouveaux ou éditer les existants en les ouvrant dans Expression Blend à l'aide du petit carré situé juste à côté de la liste déroulante.

Lorsque vous avez terminé, cliquez sur *Encode* et attendez le résultat qui devrait s'afficher dans une nouvelle fenêtre Internet explorer.

Si on regarde la page (C:\Users\VOTRE_USER\Documents\Expression\Expression Encoder\Output) qui a été générée :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">
<!-- saved from url=(0014)about:internet -->
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script type='text/javascript' src="MicrosoftAjax.js"></script>
<script type='text/javascript' src="Silverlight.js"></script>
<script type='text/javascript' src="SilverlightControl.js"></script>
<script type='text/javascript' src="SilverlightMedia.js"></script>
<script type='text/javascript' src="ExpressionPlayer.js"></script>
<script type='text/javascript' src="PlayerStrings.js"></script>
<script type='text/javascript' src="player.js"></script>
<script type='text/javascript' src="StartPlayer.js"></script>
<title></title>
<style type="text/css">
    html, body { margin: 0; padding: 0; height:100% }
    #divPlayer_0 { min-height: 100%; height:100%; }
</style>
</head>

<body style="background-color:black;margin:0,0,0,0;overflow:auto;">
    <div id="divPlayer_0">
        <script type='text/javascript'>
            var player = new StartPlayer_0();
        </script>
    </div>
</body>
</html>
```

Vous pourrez trouver tous les fichiers JavaScript en rapport dans le même dossier que le fichier *Default.html*.

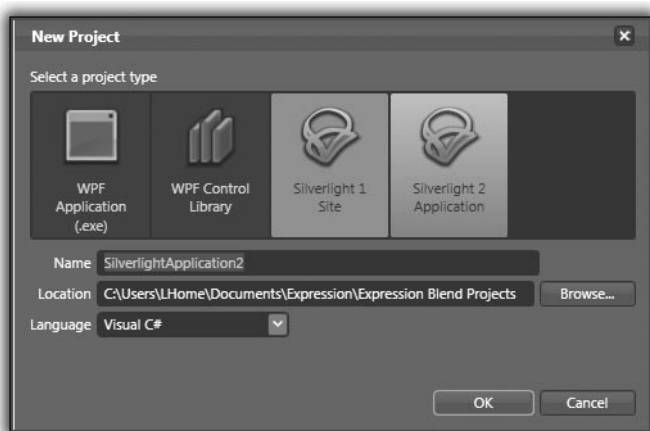
Microsoft Expression Encoder permet également de s'enregistrer (voix et son) afin de créer vos propres vidéos filmées par webcam ou caméscope. Pour cela, cliquez sur l'onglet **Live Encoding**. Une nouvelle fenêtre apparaît en *overlay*. Vous devez choisir la source de la vidéo et la source audio et cliquez sur **Start**.

Une fois l'enregistrement effectué, vous obtenez un aperçu en vous rendant dans l'onglet **output** et en cliquant sur **Launch Preview**.

2.4 Expression Blend 2

Expression Blend 2 est l'outil le plus puissant pour la réalisation d'application Silverlight 2. Pour commencer, il vous faudra télécharger le SP1 de Microsoft Expression Blend 2 afin d'avoir la possibilité de créer des projets Silverlight 2. Par défaut, Blend 2 supporte uniquement les projets Silverlight 1, c'est-à-dire les projets Silverlight à base de JavaScript.

Pour pouvez télécharger Microsoft Expression Blend 2 SP à l'adresse suivante : <http://www.microsoft.com/downloads/details.aspx?FamilyId=EB9B5C48-BA2B-4C39-A1C3-135C60BBE66&displaylang=en>.



► Fig. 2.3 :
Nouveau projet
Expression Blend 2 SP1

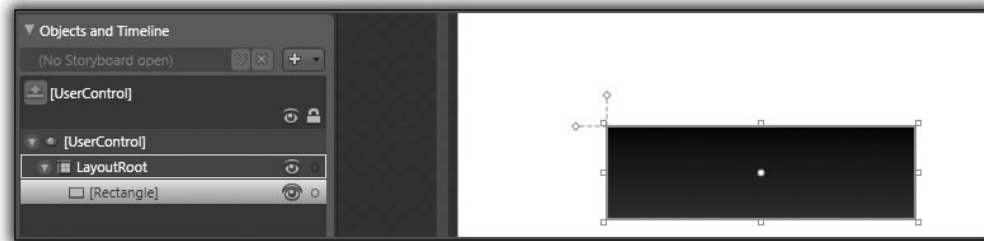
Après avoir installé le SP1, vous pouvez créer un projet Silverlight 2 en lui donnant un nom et en cliquant sur OK.

Un projet vide s'est créé. Vous pouvez basculer entre l'affiche *design* ou *XAML* par les onglets verticaux de droite.

Même un projet blanc n'est pas tout à fait vide puisque Blend vous prépare directement un conteneur de base (Grid) :

```
<UserControl
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SilverlightApplication2.Page"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot" Background="White"/>
</UserControl>
```


Si on passe à nouveau du côté *Design* et que l'on ajoute un rectangle, on remarquera que le code XAML s'est directement mis à jour :



► Fig. 2.4 : Expression Blend 2

```
<UserControl
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SilverlightApplication2.Page"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot" Background="White">
    <Rectangle Height="73" Margin="117,122,280,0"
      VerticalAlignment="Top" Stroke="#FF000000">
      <Rectangle.Fill>
        <LinearGradientBrush EndPoint="0.5,1"
          StartPoint="0.5,0">
          <GradientStop Color="#FF000000"/>
          <GradientStop Color="#FF0F2B66" Offset="1"/>
        </LinearGradientBrush>
      </Rectangle.Fill>
    </Rectangle>
  </Grid>
</UserControl>
```

Au niveau des propriétés des éléments, on retrouve assez facilement des éléments déjà vus dans *Microsoft Expression Design 2*.

Vous pouvez consulter tout les éléments XAML disponibles en cliquant sur la double flèche à gauche. Par défaut, ils ne sont pas tous affichés. Si vous voulez les afficher tous, vous devrez cocher la case en haut à droite *Show all*.

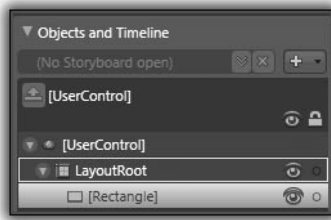


► Fig. 2.5 :
Barre des propriétés

L'élément le plus important de *Blend* est la colonne *Objects and Timeline*. Elle vous permet de visualiser les éléments que vous avez ajoutés à votre projet et surtout de voir leur situation dans le projet. En effet, cette colonne représente les éléments sous forme d'arbre de telle manière que vous puissiez voir comment les éléments sont imbriqués les uns dans les autres.

Les noms

Il est très important de donner des noms à vos éléments. C'est ainsi que vous pouvez les retrouver quand vous en avez beaucoup sur une même interface. De la même manière, lorsque vous voudrez interagir avec ceux-ci par code, vous devrez connaître leur nom dans le code. S'ils n'ont pas de noms, on peut considérer les éléments comme inaccessibles.



► Fig. 2.6 :
Objects and Timeline

C'est aussi dans cette colonne que vous allez contrôler les différentes animations de votre application. Ainsi, pour animer votre rectangle, il faut cliquer sur le petit bouton en forme de plus et donner un nom à votre animation. Vous entrez alors en mode d'enregistrement. Vous avez une ligne du temps à gauche et un encadré rouge à droite qui montre la zone d'enregistrement.

Effectuer une animation avec Blend et Silverlight s'avère très simple, surtout si vous êtes habitué à jouer avec les mêmes concepts pour *Flash/Flex*.

Tout se fait par point d'arrêt. Un point d'arrêt est fixé dès que vous déplacez un élément.

Pour créer une animation, placez-vous sur la ligne du temps et déplacez l'élément. Pour visualiser l'animation, cliquez sur le bouton **Play**.



► Fig. 2.7 : Animation dans Expression Blend 2

Ces actions très basiques génèrent du code XAML à l'arrière :

```
<UserControl
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SilverlightApplication2.Page"
  Width="640" Height="480">
  <UserControl.Resources>
    <Storyboard x:Name="Storyboard1">
      <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
        Storyboard.TargetName="rectangle">
```

```

Storyboard.TargetProperty="(UIElement.RenderTransform).
(TransformGroup.Children)[3].(TranslateTransform.Y)">
    <SplineDoubleKeyFrame KeyTime="00:00:01" Value="63"/>
    <SplineDoubleKeyFrame KeyTime="00:00:02" Value="-74"/>
</DoubleAnimationUsingKeyFrames>
<DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="rectangle"
Storyboard.TargetProperty="(UIElement.RenderTransform).
(TransformGroup.Children)[3].(TranslateTransform.X)">
    <SplineDoubleKeyFrame KeyTime="00:00:01" Value="125"/>
    <SplineDoubleKeyFrame KeyTime="00:00:02" Value="244"/>
</DoubleAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="rectangle"
Storyboard.TargetProperty="(Shape.Fill).(GradientBrush.
GradientStops)[1].(GradientStop.Color)">
    <SplineColorKeyFrame KeyTime="00:00:01"
Value="#FF0F2B66"/>
    <SplineColorKeyFrame KeyTime="00:00:02"
Value="#FFA9BCE4"/>
</ColorAnimationUsingKeyFrames>
</Storyboard>
</UserControl.Resources>

<Grid x:Name="LayoutRoot" Background="White">
    <Rectangle Height="73" Margin="117,122,280,0"
VerticalAlignment="Top" Stroke="#FF000000"
RenderTransformOrigin="0.5,0.5" x:Name="rectangle">
        <Rectangle.RenderTransform>
            <TransformGroup>
                <ScaleTransform/>
                <SkewTransform/>
                <RotateTransform/>
                <TranslateTransform/>
            </TransformGroup>
        </Rectangle.RenderTransform>
        <Rectangle.Fill>
            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                <GradientStop Color="#FF000000"/>
                <GradientStop Color="#FF0F2B66" Offset="1"/>
            </LinearGradientBrush>
        </Rectangle.Fill>
    </Rectangle>
</Grid>
</UserControl>

```

À la simple vue de ce code, vous vous rendez compte à quel point Expression Blend peut être utile.

La limite d'Expression Blend

Si les outils pouvaient générer le code sans faire d'erreurs, nous n'aurions plus besoin d'experts. Très souvent, une intervention humaine est appréciée pour optimiser le code. Tout ceci sera expliqué dans les concepts avancés. Retenez que Blend ne remplace pas un code bien écrit à la main. Il est là pour vous simplifier la vie mais tentez de toujours comprendre ce qu'il fait.

2.5 Intéraction entre Expression Blend et Visual Studio 2008

La plupart du temps, on ne commence pas un projet réel dans Expression Blend ; cela pause de nombreux problèmes de portabilité de d'organisation par la suite. Il arrive néanmoins de commencer un projet dans Expression Blend si celui-ci est destiné à donner un avant-goût du logiciel ou de l'application au client. C'est en effet un très bon moyen de convaincre un client. Vous pouvez facilement établir la comparaison avec les graphistes qui dessinent une interface et la montrer au client avant que tout soit implémenté.

Les bonnes pratiques veulent qu'on crée d'abord le projet dans Visual Studio. Si vous travaillez dans un environnement professionnel avec ces outils, vous devez utiliser une solution de gestion de version comme *Visual Source Safe* ou SVN. Malheureusement, Blend n'est pas encore compatible avec ces outils, l'utilisation de Visual Studio 2008 est donc recommandée.

Visual Studio ou Visual Developer ne proposent pas une interface d'édition du XAML aussi poussée que Expression Blend. Les prochaines versions du logiciel devraient apporter des améliorations à ce niveau. Heureusement, Microsoft a prévu l'utilisation de ces deux logiciels en simultané.

Lorsque vous créez un projet dans Visual Studio, vous avez la possibilité d'éditer les fichiers XAML directement dans Expression Blend. Pour cela, cliquez du bouton droit sur le fichier dont l'extension est *xaml*. Cela a pour effet d'ouvrir directement Blend.

La moindre modification d'un côté à des répercussions directes de l'autre. Vous êtes normalement à jour dans la gestion de vos fichiers.

Considérez toujours Visual Studio comme votre outil principal et Expression Blend comme un outil de retouche plutôt qu'un outil de création d'application Silverlight.

Expression Blend ne permet pas de modifier les fichiers contenant la logique métier. Si vous cliquez sur la petite croix devant le nom du fichier, vous verrez un sous-fichier ; c'est le fichier contenant tout ce qui est dynamique dans l'application. Seul Visual Studio ou votre Bloc-notes vous permet de modifier ces fichiers :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SilverlightApplication1
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }
    }
}
```

Si vous travaillez à deux sur un même projet, il arrive fréquemment qu'une personne travaille sur le fichier *xaml* et l'autre sur le fichier *cs*.

Attention toutefois aux abus ; la plupart du temps, pour avoir un code cohérent, il faut une interface déjà bien définie ; le fichier *xaml* est souvent plus avancé que le fichier *cs* qui lui est dédié.

Vous aurez peu l'occasion d'utiliser Expression Web et Expression Media qui ne seront pas décrits dans ce livre. Nous pouvons donc terminer ce chapitre parenthèse pour partir à la conquête des données.

2.6 Check-List

Dans ce chapitre nous avons exploré la gamme Expression avec :

- ✓ Expression Design ;
- ✓ Expression Blend ;
- ✓ Expression Encoder ;
- ✓ L'interaction entre Visual Studio et Expression Blend.

3



3.1 Utilisez SQL et votre base de données	114
3.2 Exploitez vos données sur Oracle	120
3.3 MySQL et Silverlight	124
3.4 LINQ	126
3.5 Les Web services	134
3.6 ADO.NET/Silverlight	143
3.7 Créez un widget météo	153
3.8 Traitez un flux de données RSS	174
3.9 Checklist	179

Exploiter vos sources de données

L'accès aux données est très important. Ce chapitre va vous montrer comment accéder à tout et n'importe quoi, en passant par les bases de données : que ce soit *Microsoft SQL Serveur* ou *MySQL*, ou en allant voir du côté des *Web services* qui sont de plus en plus présents. Nous montrerons par la même occasion comment exposer vos données en Web service. Nous rentrerons enfin dans un autre mode d'accès aux données : l'utilisation de *LINQ*.

3.1 Utilisez SQL et votre base de données

La majorité des données se trouve dans des bases de données. Ces dernières font partie d'un système de base de données (*SGBD*) parfois différent d'une société à l'autre. Silverlight fonctionne très bien avec *SQL Serveur*, système de base de données de Microsoft et grand concurrent d'Oracle.

D'autres *SGBD* existent comme MySQL, racheté par SUN il y a peu. La liste est longue. Ce livre couvrira l'accès en majorité à *SQL Serveur*, dont vous pouvez trouver une version gratuite et utilisable en production : *SQL Serveur Express* (cf. <http://msdn.microsoft.com/fr-fr/express/aa718378.aspx>).

Nous aurons également des exemples d'utilisations avec MySQL et Oracle qui sont deux *SGBD* fort répandus. Ainsi, vous obtiendrez une vue globale de l'exploitation des données avec Silverlight.

Silverlight, C# et SQL Serveur : introduction

Ce n'est pas Silverlight qui va accéder aux données mais bien C#, langage utilisé à l'arrière de Silverlight. Si vous n'avez aucune connaissance en C#, nous vous conseillons la lecture de l'annexe 2, *Introduction au C#*. Autrement, vous ne comprendrez pas ce qui va suivre.

Nous allons voir comment utiliser *SQL Serveur*. En C#, il existe plusieurs moyens d'effectuer des requêtes sur *SQL Serveur*. Pour ce faire, nous avons besoin des éléments suivants :

- une chaîne de connexions permettant de vous authentifier et de cibler la base de données à attaquer ;
- une connexion qui s'ouvre grâce aux paramètres placés dans la chaîne de connexion ;
- une commande qui est une chaîne de caractères représentant une requête *SQL* ;
- un objet pour récupérer les résultats afin de pouvoir les afficher.

Une fois que nous avons tous ces éléments, nous pouvons commencer.

Tout d'abord, la chaîne de connexion. C'est un concept qui reviendra pour les autres *SGBD* également. Pour *SQL Serveur*, elle ressemble à ceci :

```
Data Source=PC-MACWIN\SQLEXPRESS;Initial
Catalog=WipusBD;Persist Security Info=True;User ID=WipusDBManager;
➡ Password=WipusManager
```

Cette chaîne de connexion, nous devons la sauvgarder dans une variable de type string. Ce type de variable est spécialement prévu pour accepter les chaînes de caractères :

```
string connectionString = @"Data
Source=PC-MACWIN\SQLEXPRESS;Initial
Catalog=WipusBD;Persist Security Info=True;User ID=WipusDBManager;
➡ Password=WipusManager";
```

Une fois cette chaîne sauvegardée, nous pouvons l'utiliser pour initialiser une connexion. Cela se fait à l'aide de l'objet SqlConnection :

```
SqlConnection myConnection;
myConnection = new SqlConnection(connectionString);
```

Une fois la connexion initialisée, nous devons l'ouvrir :

```
try
{
    myConnection.Open();
}
catch (Exception e)
{
    Console.WriteLine(e.ToString());
}
```

Le try/catch permet de détecter s'il y a eu une erreur lors de la connexion à la base de donnée. Si c'est le cas, vous pourrez la gérer dans le catch. Autrement, le code continuera à s'exécuter sans passer dans le catch.

Avant d'aller plus loin, nous allons procéder à un rappel concernant SQL.

SQL

Structured Query Language est un langage qui vous permet d'accéder aux données situées dans une base de données. C'est un langage uniformisé qui doit, normalement, fonctionner sur tous les *SGBD*. Cependant, vous trouverez quelques nuances quand vous utiliserez SQL Server ou Oracle, ces deux *SGBD* étant si puissants qu'ils ont ajouté un lot de nouvelles fonctionnalités au *SQL*.

Sans rentrer dans les détails, *SQL* permet de réaliser 4 actions basiques :

- sélectionner ;
- supprimer ;
- éditer ;
- ajouter.

Chaque action que peut effectuer SQL a son mot-clé. Ainsi, pour la sélection, on a le mot-clé SELECT, pour la suppression, le mot-clé est DELETE, pour éditer UPDATE et INSERT pour l'ajout dans une base de données.

Pour la sélection, outre le mot-clé SELECT, nous devons signifier ce que nous souhaitons sélectionner (la projection) et le critère de sélection sur cette sélection.

Par exemple : *Rechercher le nom, le prénom et l'âge des étudiants de deuxième année.*

Nous avons ici une projection : nom, prénom et âge, ainsi qu'une sélection, c'est-à-dire le fait de sélectionner uniquement les élèves de deuxième année.

Pour la projection, nous avons presque tous les éléments pour notre requête :

```
SELECT nom, prenom, age
FROM etudiants
```

Pour la sélection, nous devons introduire un nouveau mot-clé : WHERE :

```
SELECT nom, prenom, age
FROM etudiants
WHERE annee = 2
```

Dans cet exemple, nous ne prenons en compte une seule table. Il arrive fréquemment de devoir aller rechercher une information située dans plusieurs tables. Imaginez que notre base de données stocke le cours auquel l'étudiant participe actuellement. Pour cela, il y aurait une autre table *cours* avec une référence vers la table des étudiants. Si vous voulez la liste des étudiants se trouvant au cours de maths, nous allons obtenir :

```
SELECT etudiants.nom, etudiants.prenom, etudiants.age
FROM etudiants, cours
WHERE etudiants.Id = cours.Id
AND cours.Nom = "Math"
```

Cette façon ne représente qu'une manière de chercher le résultat. Nous pourrions utiliser d'autres méthodes avec les mots-clés INNER JOIN ou JOIN.

Pour ce qui est de l'insertion de nouveaux éléments ainsi que de la suppression et modification, vous découvrirez, au long du chapitre, la syntaxe fort proche de la sélection.

Les commandes SQL en C#

Après cette petite parenthèse sur SQL, nous pouvons continuer avec deux des quatre éléments de base dont nous avons besoin pour effectuer sur requête sur SQL Serveur. Cela se fait à l'aide de l'objet SqlCommand :

```
SqlCommand cmd = new SqlCommand("SELECT nom, prenom, age " +  
    "FROM etudiants " +  
    "WHERE annee = 2 " +  
    "AND age < 22 ", con);
```

L'objet `SqlCommand` prend en paramètre une chaîne de caractères représentant la requête *SQL* ainsi que l'objet connexion que nous avons initialisé auparavant (`SqlConnection`).

Il peut arriver que la requête ait des données dites dynamiques. Imaginez par exemple que dans votre Silverlight, vous ayez à un moment donné demandé l'âge maximal des élèves que vous voudriez afficher. Cette donnée aura été sauvée dans une variable de type `int` ou `short` vu que l'âge d'une personne dépasse rarement les 100 ans. Vous pouvez exploiter ces données au sein de votre requête :

```
SqlCommand cmd = new SqlCommand("SELECT nom, prenom, age " +  
    "FROM etudiants " +  
    "WHERE annee = " +  
    "AND age < " + VOTRE VARIABLE ICI, myConnection);
```



Attention aux hackers

En utilisant cette méthode, vous ne vous mettez pas à l'abri des hackers qui pourraient effectuer une injection SQL.

Une injection SQL est une injection de code via un formulaire qui a pour objectif de détourner votre requête de son but premier dans l'espoir de corrompre votre système de base de données.

Pour éviter cela, vous ne devez jamais placer de paramètre directement dans la chaîne de caractères mais préférer l'utilisation de l'objet `SqlParameter`.

`SqlParameter` est un objet qui définit un paramètre de votre requête. Le paramètre est désigné par un identifiant qui mappe les paramètres de la requête *SQL* :

```
SqlCommand cmd = new SqlCommand("SELECT count(*) " +  
    "FROM video " +  
    "WHERE Status = " +  
    "1 AND UserId = @userId " +  
    "AND IdVideo = @IdVideo ", con);  
  
SqlParameter pVideoId =  
    new SqlParameter("IdVideo", SqlDbType.BigInt);  
pVideoId.Value = VideoId;  
cmd.Parameters.Add(pVideoId);  
MembershipUser user = Membership.GetUser();  
SqlParameter pUserId =  
    new SqlParameter("UserId", SqlDbType.UniqueIdentifier);
```

```
pUserId.Value = UserId;
cmd.Parameters.Add(pUserId);
```

Ainsi, les attaques *SQL* ne sont pas possibles. Comme on connaît le type qu'on doit obtenir, on réduit déjà le champ de l'attaque. De plus, il traite automatiquement les caractères tels que `"`.

Pour exécuter notre requête, nous devons créer un *reader*. Cet objet va nous permettre de lire les résultats de la requête pour les afficher ensuite :

```
private static void ReadOrderData(string connectionString)
{
    string queryString =
        "SELECT OrderID, CustomerID FROM dbo.Orders;";
    using (SqlConnection connection = new SqlConnection(
        connectionString))
    {
        SqlCommand command = new SqlCommand(
            queryString, connection);
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        try
        {
            while (reader.Read())
            {
                Console.WriteLine(String.Format("{0}, {1}",
                    reader[0], reader[1]));
            }
        }
        finally
        {
            // Always call Close when done reading.
            reader.Close();
        }
    }
}
```

Exemple

Cet exemple provient d'une application console. Étant donné que vous n'avez pas encore les notions suffisantes pour ASP.NET, nous avons volontairement limité l'exemple.

Rendez-vous au chapitre 5, *Silverlight et ASP.NET*, pour voir comment traiter les données correctement. Plus loin dans ce chapitre, nous découvrirons également comment fonctionne l'accès direct à une base de données via Silverlight.

Les méthodes montrées jusqu'ici ne sont applicables qu'au travers d'un Web service. Nous verrons les Web services plus loin dans ce chapitre. Néanmoins, il est rare qu'un Web service expose une méthode qui renvoie un `DataReader`. Vous pouvez utiliser la fonction suivante :

```
/// <summary>
///   Converts a SqlDataReader to a DataSet
///   <param name='reader'>
///   SqlDataReader to convert.</param>
///   <returns>
///   DataSet filled with the contents of the reader.</returns>
///   </summary>
public static DataSet convertDataReaderToDataSet(SqlDataReader reader)
{
    DataSet dataSet = new DataSet();
    do
    {
        // Create new data table

        DataTable schemaTable = reader.GetSchemaTable();
        DataTable dataTable = new DataTable();

        if (schemaTable != null)
        {
            // A query returning records was executed

            for (int i = 0; i < schemaTable.Rows.Count; i++)
            {
                DataRow dataRow = schemaTable.Rows[i];
                // Create a column name that is unique
                // in the data table
                string columnName = (string)dataRow["ColumnName"];
                //+ "<C" + i + ">";
                // Add the column definition to the data table
                DataColumn column = new DataColumn(columnName,
                    (Type)dataRow["DataType"]);
                dataTable.Columns.Add(column);
            }

            dataSet.Tables.Add(dataTable);

            // Fill the data table we just created

            while (reader.Read())
            {
                DataRow dataRow = dataTable.NewRow();
```

```

        for (int i = 0; i < reader.FieldCount; i++)
            dataRow[i] = reader.GetValue(i);

        dataTable.Rows.Add(dataRow);
    }
}
else
{
    // No records were returned

    DataColumn column = new DataColumn("RowsAffected");
    dataTable.Columns.Add(column);
    dataSet.Tables.Add(dataTable);
    DataRow dataRow = dataTable.NewRow();
    dataRow[0] = reader.RecordsAffected;
    dataTable.Rows.Add(dataRow);
}
}
while (reader.NextResult());
return dataSet;
}

```

3.2 Exploitez vos données sur Oracle

Oracle est une base de données reconnue au niveau mondiale. Elle est utilisée par les plus grandes sociétés. C'est un concurrent de SQL Serveur.

Elle permet d'être configurée pour supporter de grosse consommation de données.

Le but de cette section n'est pas d'apprendre à utiliser Oracle mais de réussir à exploiter les données d'Oracle directement en C#. Il vous faudra par la suite créer un Web service et attaquer votre Web service dans votre application Silverlight pour pouvoir récolter les données.

Avant de commencer, vous devez installer les outils Oracle pour Visual Studio 2008 disponibles à cette adresse : <http://www.oracle.com/technology/software/tech/windows/odpnet/>.

Vous pouvez également télécharger *ODP.NET (Oracle Data provider)* qui sera utilisé dans cette section. Vous le trouverez à l'adresse suivante : <http://www.oracle.com/technology/software/tech/windows/odpnet/index.html>.

Dans votre projet *.NET*, n'oubliez pas d'ajouter la référence à la DLL d'Oracle :

```
using Oracle.DataAccess.Client; // C#
```


Vous pouvez ensuite vous connecter à la base de données *Oracle* via sa chaîne de connexion qui doit ressembler à ceci :

```
OraDb=
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS= (PROTOCOL=TCP) (HOST=OTNSRVR) (PORT=1521))
    )
    (CONNECT_DATA=
      (SERVER=DEDICATED)
      (SERVICE_NAME=ORCL)
    )
  )
```

Il faut donc placer cette chaîne de connexion dans une variable afin de pouvoir l'utiliser plus tard dans le programme :

```
string oradb = "Data Source=(DESCRIPTION="
  + " (ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP) (HOST=ORASRVR) (PORT=1521))) "
  + " (CONNECT_DATA=(SERVER=DEDICATED) (SERVICE_NAME=ORCL))) ";
  + "User Id=scott;Password=tiger;" ;
```

Une fois la chaîne sauvee, vous pouvez créer une connexion à partir de cette chaîne de la manière suivante :

```
OracleConnection conn = new OracleConnection(oradb);
```

Ou si vous préférez :

```
OracleConnection conn = new OracleConnection();
conn.ConnectionString = oradb;
```

Et comme vous le feriez pour SQL Serveur, ouvrir la connexion :

```
conn.Open();
```

La suite est pratiquement la même que ce que vous connaissez avec SQL Serveur. Vous devez créer une commande :

```
string sql = "select dname from dept where deptno = 10"; // C#
OracleCommand cmd = new OracleCommand(sql, conn);
cmd.CommandType = CommandType.Text;
```

Et utiliser un *Reader* pour aller rechercher les résultats :

```
OracleDataReader dr = cmd.ExecuteReader();
dr.Read();

label1.Text = dr["dname"].ToString();
// C# retrieve by column name
label1.Text = dr.GetString(0).ToString();
// return a .NET data type
label1.Text = dr.GetOracleString(0).ToString();
// return an Oracle data type
```

On peut alors prendre les valeurs du *Reader* :

```
label1.Text = dr.GetInt16("deptno").ToString();
```

Et terminer la connexion proprement, comme nous l'aurions fait avec SQL Serveur :

```
conn.Close();
conn.Dispose();
```

Voici une meilleure façon de faire lorsqu'on voit le code dans son ensemble :

```
using (OracleConnection conn = new OracleConnection(oradb))
{
    conn.Open();

    OracleCommand cmd = new OracleCommand();
    cmd.Connection = conn;
    cmd.CommandText =
        "select dname from dept where deptno = 10";
    cmd.CommandType = CommandType.Text;

    OracleDataReader dr = cmd.ExecuteReader();
    dr.Read();

    label1.Text = dr.GetString(0);
}
```



Gestion d'erreur

Lorsque vous programmez, n'oubliez jamais de traiter correctement les erreurs, notamment l'ouverture d'une connexion.

Pour gérer efficacement les erreurs, utilisez try catch :

```
try
{
    conn.Open();

    OracleCommand cmd = new OracleCommand();
    cmd.Connection = conn;
    cmd.CommandText = "select dname from dept where deptno = " +
        ➤ textBox1.Text;
    cmd.CommandType = CommandType.Text;

    if (dr.Read()) // C#
    {
        label1.Text = dr["dname"].ToString();
        // or use dr.GetOracleString(0).ToString()
    }
}
catch (Exception ex) // catches any error
{
    MessageBox.Show(ex.Message.ToString());
}
finally
{
    // In a real application, put cleanup code here.
}
```

Le *provider* Oracle fournit une série d'erreurs que vous pouvez interpréter à l'aide de l'objet `OracleException` :

```
catch (OracleException ex) // catches only Oracle errors
{
    switch (ex.Number)
    {
        case 1:
            MessageBox.Show("Corruption clef primaire");
            break;
        case 12545:
            MessageBox.Show("La base de données ne répond pas.");
            break;
        default:
            MessageBox.Show("Autre erreur: " + ex.Message.ToString());
            break;
    }
}
catch (Exception ex) // catches any error
```

```
{
    MessageBox.Show(ex.Message.ToString());
}
```

Si vous effectuez une requête qui rapatrie plusieurs éléments, utilisez la méthode Read du `DataReader` :

```
while (dr.Read()) // C#
{
    listBox1.Items.Add("The " +
        dr["dname"].ToString() +
        " department is in " +
        dr["loc"].ToString());
}
```

Voilà notre introduction à l'accès aux données via Oracle terminée. Il y a bien entendu beaucoup plus de choses à faire avec cette *DLL* mais ce n'est pas l'objectif de notre livre.

Reportez-vous à l'annexe 3, Webographie, pour obtenir les liens qui vous guideront vers de l'information sur le sujet.

3.3 MySQL et Silverlight

MySQL est une base de données très répandue. Elle est souvent combinée à des projets PHP. Si vous êtes habitué à PHP et MySQL, nous vous conseillons la lecture du tutorial suivant : <http://nico-pyright.developpez.com/tutorial/vs2008/csharp/silverlightandmysql/>.

Sur le site de MySQL, vous trouverez assez facilement un provider pour *.NET*.

MySQL n'est pas fort différent d'Oracle et SQL Serveur dans la manière de rechercher les données. Le nom des objets va vite vous donner une idée de son fonctionnement :

```
private MySqlConnection conn;
private DataTable data;
private MySqlDataAdapter da;
private System.Windows.Forms.DataGrid dataGrid;
private MySqlCommandBuilder cb;
```

Nous avons également besoin d'une chaîne de connexion :

```
string connStr = String.Format("server={0};user " +
    "id={1}; password={2}; database=mysql; pooling=false",
    server.Text, userid.Text, password.Text );
```

La chaîne de connexion est plus petite que pour Oracle et SQL Serveur. C'est quasiment la seule chose qui changera dans notre code. Après avoir enregistré cette chaîne de connexion, nous pouvons l'ouvrir :

```
try
{
    conn = new MySqlConnection( connStr );
    conn.Open();

    GetDatabases();
}
catch (MySqlException ex)
{
    MessageBox.Show( "Error connecting to the server: " + ex.Message );
}
```

Nous pouvons ensuite déclarer une commande et l'exécuter :

```
MySqlDataReader reader = null;

conn.ChangeDatabase( databaseList.SelectedItem.ToString() );

MySqlCommand cmd = new MySqlCommand("SHOW TABLES", conn);
try
{
    reader = cmd.ExecuteReader();
    tables.Items.Clear();
    while (reader.Read())
    {
        // traitement
    }
}
catch (MySqlException ex)
{
    MessageBox.Show("Failed to populate table list: " + ex.Message );
}
finally
{
    if (reader != null) reader.Close();
}
```

Nous avons vu ici comment changer la base de données concernée par la requête, ce qui ne peut pas se faire dans les autres *providers*. Vous devrez, à chaque fois, utiliser une chaîne de connexion différente pour chaque base de données.

Vous avez aussi la possibilité d'utiliser des `DataTable` :

```
data = new DataTable();

da = new MySqlDataAdapter("SELECT * FROM " +
    tables.SelectedItem.ToString(), conn );
cb = new MySqlCommandBuilder( da );

da.Fill( data );

dataGridView.DataSource = data;
```

C'est à peu près tout ce que vous devez connaître sur les connexions à un serveur *MySQL*.

3.4 LINQ

LINQ est apparu avec le Framework 3.0 de la plateforme .NET. Il a vite été apprécié par les développeurs pour la facilité qu'il apporte et sa merveilleuse intégration dans Visual Studio 2008.

LINQ a su plaire aux développeurs qui avaient l'habitude d'aller rechercher leurs données dans des bases de données : *SQL Serveur* dans un premier temps. Il existe maintenant profusion de providers qui permettent de connecter d'autres *SGBD*. En effet, même si le SQL est un langage de choix, il n'existe pas de compilateur qui sache repérer une erreur dans notre syntaxe SQL. Il n'existe pas non plus d'IntelliSense pour SQL.

Ce n'est évidemment pas la seule raison de ce succès. *LINQ* est très facile à prendre en main. En effet, sa syntaxe est simple et ressemble beaucoup au SQL. Voyez par vous-même l'exemple d'un projet *LINQ to Object* :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LINQ
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] names = { "Burke", "Connor", "Frank",
                               "Everett", "Albert", "George",
                               "Harris", "David" };
        }
    }
}
```

```

        IEnumerable<string> query = from s in names
                                    where s.Length == 5
                                    orderby s
                                    select s.ToUpper();

        foreach (string item in query)
            Console.WriteLine(item);

        Console.ReadLine();
    }
}
}

```

Nous retrouvons les différents éléments qui constituent une requête SQL :

- *From* ;
- *Where* ;
- *Order by*, etc.

Il existe de nombreux projets LINQ. La communauté *.NET* s'est emparée du projet pour l'étendre à d'autres sources de données. Voici un tableau récapitulatif et non exhaustif des différents projets LINQ :

 Tableau 3.1 : Liste des providers LINQ

Provider	Description
LINQ to Object	Permet d'utiliser LINQ sur des collections d'objets.
LINQ to SQL	Permet d'utiliser LINQ avec une source de données relationnelles (SQL Serveur pour le moment).
LINQ to XML	Utilisé pour manipuler des sources XML
LINQ to DataSet	Un provider pour utiliser LINQ avec des DataSets
LINQ to Entities	LINQ s'appuyant sur un modèle d'entité
LINQ to Amazon	Providers pour attaquer les données d'Amazon
LINQ to Active Directory	Providers pour contacter Active Directory
LINQ to Bindable Sources	Permet de faciliter la gestion des états d'une application.
LINQ over C#	Permet de passer du C# à l'aide de LINQ.
LINQ to CRM	Providers pour CRM

 Tableau 3.1 : Liste des providers LINQ

LINQ to Geo	LINQ pour manipuler des données géo spatiales
LINQ to Excel	LINQ pour manipuler des tableaux Excel
LINQ to Expressions	Permet de manipuler des arbres en utilisant LINQ.
LINQ to FlickrR	Permet d'aller rechercher des informations sur un compte Flickr.
LINQ to Google	LINQ pour requêter Google
LINQ to Indexes	Permet d'ajouter la notion d'index à LINQ.
LINQ to JSON	Pour manipuler JSON à l'aide de LINQ
LINQ to NHibernate	Permet de faire le pont entre LINQ et NHibernate.
LINQ to JavaScript	Permet d'utiliser LINQ dans du JavaScript.
LINQ to LDAP	Permet de manipuler un annuaire LDAP avec LINQ.
LINQ to LBLGen Pro	Permet d'étendre LINQ to SQL (prise en charge de plus de fonctionnalités venant de SQL).
LINQ to Lucene	LINQ pour la manipulation de chaînes de caractères
LINQ to Metaweb	LINQ pour Freebase
LINQ to MySQL, Oracle et PostGreSql	LINQ pour Oracle, MySQL et PostGreSql (implémentation de LINQ to SQL 2)
LINQ to NCover	LINQ pour NCover
LINQ to Opf3	LINQ pour le Framework Opf3
LINQ to Parallel	Aussi appelé PLINQ, permet de programmer en multi core avec LINQ.
LINQ to RDF File	Comme son nom l'indique, permet de manipuler des fichiers RDF à l'aide de LINQ.
LINQ to SharePoint	Permet de manipuler les listes SharePoint avec LINQ.
LINQ to SimpleDB	LINQ to SQL pour Amazon SimpleDB
LINQ to Streams	Manipulation de données de streaming avec LINQ
LINQ to WebQueries	Permet de traiter le Web comme une base de données.
LINQ to WMI	LINQ pour WMI
LINQ to XtraGrid	LINQ pour manipuler les grids

Le tableau parle de lui-même, il existe à ce jour énormément de *providers* pour LINQ. Nous n'allons pas les détailler tous dans ce livre.

LINQ, un peu d'explication

Pour comprendre comment fonctionne *LINQ*, nous devons reprendre notre exemple et le décomposer :

```
IEnumerable<string> query = from s in names
                             where s.Length == 5
                             orderby s
                             select s.ToUpper();
```

La première chose qui choque, lorsque nous rencontrons une requête *LINQ*, ce sont les nouveaux mot-clés :

- from ;
- where ;
- orderby ;
- select, etc.

Ces mots-clés ont été introduits dans le Framework 3.0. Lorsque le compilateur tombe sur ces mots-clés, il transforme tout en succession de méthodes :

```
IEnumerable<string> query = names
    .Where(s => s.Length == 5)
    .OrderBy(s => s)
    .Select(s => s.ToUpper());
```

Nous n'allons pas nous éterniser sur le sujet tant il est vaste. Nous verrons juste un exemple d'utilisation de *LINQ to XML* qui nous permettra de manipuler du XAML.

LINQ to XML par l'exemple

Notre exemple est assez basique. Récupérer du XAML d'un Web service et l'utiliser avec *LINQ to XML* pour l'injecter dynamiquement dans notre code. C'est un exemple assez courant lorsqu'on apprend *ASP.NET* ; nous allons ici l'adapter à Silverlight.

Nous créerons donc une application Silverlight munie uniquement d'un `StackPanel`. Dans cet élément, nous viendrons charger dynamiquement du contenu :

```
<UserControl x:Class="HDI_Silverlight_LinqandXaml_sl_cs.Page"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="400" Height="300" Loaded="UserControl_Loaded">
<Grid x:Name="LayoutRoot" Background="White">
    <StackPanel VerticalAlignment="Top" x:Name="stkMenu"
        Orientation="Horizontal"></StackPanel>
</Grid>
</UserControl>

```

Dans le code attaché à notre fichier XAML, nous allons instancier un objet qui se connectera au Web service que nous verrons plus tard :

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Xml.Linq;

namespace HDI_Silverlight_LinqandXaml_sl_cs
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }

        private void UserControl_Loaded(object sender, RoutedEventArgs e)
        {
            MenuServiceReference.MenuServiceClient msc =
                new MenuServiceReference.MenuServiceClient();

            msc.GetMenuItemsCompleted +=
                new EventHandler<MenuServiceReference.GetMenuItems
                    CompletedEventArgs>
                    (msc_GetMenuItemsCompleted);

            msc.GetMenuItemsAsync();
        }

        void msc_GetMenuItemsCompleted(object sender,
            MenuServiceReference.GetMenuItemsCompletedEventArgs e)
        {
            foreach (XElement xe in e.Result)
            {
                stkMenu.Children.Add(System.Windows.Markup.XamlReader.Load
                    (xe.ToString()) as UIElement);
            }
        }
    }
}

```

```

    }
}

```

Toujours le même fonctionnement. On va rechercher le contenu et on appelle une méthode une fois le contenu téléchargé.

Si on regarde au niveau de notre Web service, nous avons une interface :

```

using System.Collections.Generic;
using System.ServiceModel;
using System.Xml.Linq;

namespace HDI_Silverlight_LinqandXaml_web_cs
{
    [ServiceContract]
    public interface IMenuService
    {
        [OperationContract]
        List<XElement> GetMenuItems();
    }
}

```

Cette interface ne définit qu'une méthode. Cette méthode est complétée dans la classe qui implémente l'interface :

```

using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace HDI_Silverlight_LinqandXaml_web_cs
{
    public class MenuService : IMenuService
    {
        public List<XElement> GetMenuItems()
        {
            List<XElement> menuelementlist = new List<XElement>();
            XNamespace xmlns =
                "http://schemas.microsoft.com/client/2007";

            var elements =
                from x in DataBaseClass.GetDataElements()
                select new XElement(xmlns + x.ElementType,
                    new XAttribute("Width", x.Width),
                    new XAttribute("Height", x.Height),
                    new XAttribute("Content", x.Content));
        }
    }
}

```

```

        menuelementlist = elements.ToList<XElement>();

        return menuelementlist;
    }
}

```

C'est dans ce fichier que l'on voit l'utilisation de *LINQ to XML*. *LINQ* va rechercher les données dans une classe *DataBaseClass* :

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace HDI_Silverlight_LinqandXaml_web_cs
{
    public static class DataBaseClass
    {
        public static List<DataElement> GetDataElements()
        {
            List<DataElement> lst = new List<DataElement>();
            lst.Add(new DataElement { ElementType = "Button",
                                      Width = "100",
                                      Height = "50",
                                      Content = "Menu Item 1" });
            lst.Add(new DataElement { ElementType = "Button",
                                      Width = "150",
                                      Height = "50",
                                      Content = "Menu Item 2" });
            lst.Add(new DataElement { ElementType = "Button",
                                      Width = "100",
                                      Height = "50",
                                      Content = "Menu Item 3" });

            return lst;
        }
    }

    public class DataElement
    {
        public string ElementType { get; set; }
        public string Width { get; set; }
        public string Height { get; set; }
        public string Content { get; set; }
    }
}

```

Cette classe fait office de base de données. On aurait pu imaginer aller rechercher l'objet dans une base de données.

Une fois le Web service terminé, nous devons ajouter une référence dans notre projet Silverlight, ce qui crée un fichier *ServiceReference.config* :

```
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IMenuService"
          maxBufferSize="2147483647"
          maxReceivedMessageSize="2147483647">
          <security mode="None" />
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://localhost:7107/MenuService.svc"
        binding="basicHttpBinding"
        bindingConfiguration="BasicHttpBinding_IMenuService"
        contract="MenuServiceReference.IMenuService"
        name="BasicHttpBinding_IMenuService" />
    </client>
  </system.serviceModel>
</configuration>
```

L'exemple est terminé. Il vous reste à déclarer un tag *Silverlight* dans une page pour tester votre code :

```
<%@ Page Language="C#" AutoEventWireup="true" %>

<%@ Register Assembly="System.Web.Silverlight"
Namespace="System.Web.UI.SilverlightControls"
TagPrefix="asp" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%;">
<head runat="server">
  <title>HDI-Silverlight-LinqandXaml-sl-cs</title>
</head>
<body style="height:100%;margin:0;">
  <form id="form1" runat="server" style="height:100%;">
    <asp:ScriptManager ID="ScriptManager1"
      runat="server"></asp:ScriptManager>
```

```

<div style="height:100%;">
  <asp:Silverlight ID="Xaml1" runat="server"
    Source="~/ClientBin/HDI-Silverlight-LinqandXaml-sl-cs.xap"
    MinimumVersion="2.0.31005.0" Width="100%"
    Height="100%" />
</div>
</form>
</body>
</html>

```

Nous pouvons maintenant passer à la partie Web service qui vous permettra de mieux comprendre comment les exemples précédents fonctionnent.

LINQ est très complexe et le Framework a du connaître beaucoup de modification et d'ajout pour permettre l'utilisation de LINQ.

Reportez-vous à l'annexe 3, Webographie, si vous souhaitez plus d'information sur l'utilisation de LINQ.

Aujourd'hui, le provider LINQ le plus utilisé est *LINQ to Entities*. Il vous permet de vous connecter à votre base de données et traiter les données de celle-ci comme si s'agissait d'objets que vous avez l'habitude de manipuler dans vos programmes.

3.5 Les Web services

Les Web services sont apparus avec le Web 2.0. C'est l'une des meilleures choses qui soit arrivé au Web. Ils permettent d'exposer des données dans un format standard afin que tout le monde puisse les utiliser. C'est l'optique du Web 2.0 ; celle du partage dans tous les sens du terme.

On voit pourtant encore trop de sites web se fermer sur eux-mêmes alors qu'une bonne API permet de créer un système d'applications qui gravite autour. C'est le cas de *Twitter* ou encore *Flickr* qui grâce à leur API se voient utiliser par de nombreuses autres applications qui ajoutent des fonctionnalités à un service déjà existant.

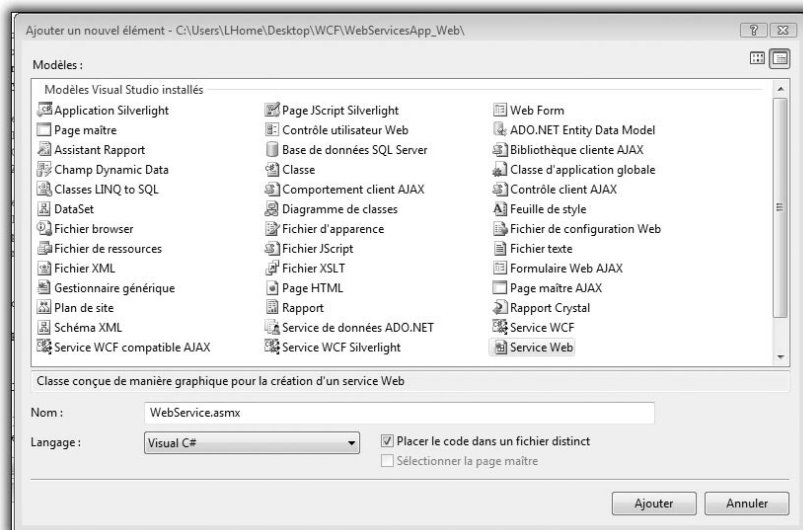
Vient le problème de rentabilité. Exposer des données pour que d'autres les consommes coûte énormément d'argent. Microsoft propose donc certain de ces Web services payants, comme *Live Earth*, pour des utilisations à grande échelle.

Ce qui est certain, c'est que le modèle économique du Web 2.0 n'est pas encore très clair. Peu importe, ces Web services sont là et il faut que vous sachiez les exploiter dans vos applications Silverlight. Voire même créer vous-même vos Web services pour consommer vos données plus facilement.

Le *Framework .NET* propose plusieurs types de Web services. Pour comprendre, il faut un petit historique. En fait, avant l'arrivée de Silverlight et du *Framework 3.0*, les Web services se créaient avec ASP.NET sous forme de fichier *asmx*. Depuis, Microsoft a introduit *WCF* (*Windows Communication Foundation*). Cette nouvelle brique du *Framework* permet de créer des Web services respectant des normes qui ne sont plus seulement les normes de Microsoft mais des normes appliquées un peu partout sur le Web. Ce qui a pour but de créer un Web interopérable, c'est-à-dire où la technologie utilisée n'influence plus les données qu'elle peut manipuler. Ainsi, si *WCF* expose des données, du code *PHP* sera capable d'aller rechercher l'information, etc. De la même manière, du code *.NET* pourra récupérer de l'information en provenance de *PHP* ou *Ruby*. C'est ce qui se passe lorsque vous contactez l'*API Twitter* puisque *Twitter* et son *API* sont développés en *Ruby* et pourtant, *.NET* sait facilement lire les données exposées sur l'*API*.

Dans l'exemple qui va nous servir d'apprentissage, nous allons créer deux Web services différents. Un en *ASP.NET*, même si vous n'avez pas encore eu le temps de lire le chapitre d'introduction. Un autre, en *WCF*, qui exposera des données simples et complexes, que nous devrons exploiter dans notre application Silverlight.

Vous devez donc créer une application Silverlight dans Visual Studio. Visual Studio vous demande ensuite si vous souhaitez attacher un projet *ASP.NET* à votre projet Silverlight. C'est le cas ici. Dans ce projet *ASP.NET*, nous devons ajouter un fichier *ASMX* (fichier de Web service) :



► Fig. 3.1 : Création d'un Web service

i Web service WCF

Sur la même image, on remarque la présence d'un fichier *WCF*. Lorsque vous créez un fichier *SVC* (*Web service WCF*), vous devrez revenir sur le même écran et sélectionner *Web service WCF*.

Vous avez donc un nouveau fichier dans votre projet :

```
<%@ WebService Language="C#"
    CodeBehind="~/App_Code/SimpleAsmx.cs"
    Class="SimpleAsmx" %>
```

Ce fichier fait référence à un autre fichier situé dans *App_Code*. Si ce fichier n'existe pas, créez-le. Ce fichier définira les différentes méthodes que notre Web service expose :

```
using System;
using System.Collections;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

/// <summary>
/// Summary description for SimpleAsmx
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding
    (ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web service to be called from script,
// using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class SimpleAsmx
    : System.Web.Services.WebService
{

    public SimpleAsmx()
    {

        //Uncomment the following line if using
        //designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld()
    {
```



```

        return "Hello World";
    }

    [WebMethod]
    public string SayHelloToMe(string name)
    {
        return string.Format("Hello {0}, how are you?", name);
    }
}

```

Nous avons ici deux méthodes exposées. Pour exposer une méthode, il suffit de lui appliquer un attribut `WebMethod`. La première fonction fait un célèbre *Hello World* et la deuxième prend un paramètre et demande à l'utilisateur comment il va.

Notre Web service *ASP.NET* est terminé. Nous pouvons faire la même chose pour *WCF*. De la même manière, vous allez voir apparaître dans votre projet un nouveau fichier dont l'extension est `svc` :

```

<%@ ServiceHost Language="C#"
    Debug="true"
    Service="SimpleWCF"
    CodeBehind="~/App_Code/SimpleWCF.cs" %>

```

Ce fichier fait référence à un fichier code dans `App_Code` comme pour le Web service *ASP.NET* :

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

// NOTE: If you change the class name "SimpleWCF"
// here, you must also update the reference to
// "SimpleWCF" in Web.config.
public class SimpleWCF : ISimpleWCF
{
    #region ISimpleWCF Members

    public string SayHelloToMe(string name)
    {
        return string.Format("Hello {0}, how are you today?", name);
    }
}

```

```

#endregion

#region ISimpleWCF Members

public List<Person> GetPeople()
{
    List<Person> ppl = new List<Person>();

    ppl.Add(new Person() { FirstName = "Tim", LastName = "Heuer"
    ➤ });
    ppl.Add(new Person() { FirstName = "Zane", LastName = "Heuer"
    ➤ });
    ppl.Add(new Person() { FirstName = "Steve", LastName =
    ➤ "Ballmer" });

    return ppl;
}

#endregion
}

```

Nous avons ici deux méthodes. L'une renvoie un texte et l'autre, une série de personnes. Ici, pas d'attributs. Tout se trouve dans l'interface implémentée par notre classe *SimpleWCF* :

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

[ServiceContract]
public interface ISimpleWCF
{
    [OperationContract]
    string SayHelloToMe(string name);

    [OperationContract]
    List<Person> GetPeople();
}

[DataContract]
public class Person
{
    [DataMember]

```

```

public string FirstName;

[DataMember]
public string LastName;
}

```

Du côté de notre Silverlight, nous n'avons pas énormément de chose. Juste un *textbox* pour donner son nom et un *textblock* pour afficher les résultats. Un bouton se trouve également dans notre *XAML* pour demander l'accès au Web service :

```

<UserControl x:Class="WebServicesApp.Page"
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <StackPanel x:Name="OurStack" Orientation="Vertical">
            <TextBox x:Name="InputText"></TextBox>
            <TextBlock x:Name="OutputText"></TextBlock>
            <ItemsControl x:Name="PeopleList">
                <ItemsControl.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding FirstName}" />
                    </DataTemplate>
                </ItemsControl.ItemTemplate>
            </ItemsControl>
            <Button x:Name="CallServiceButton"
                Content="Call WCF Simple"
                Click="CallServiceButton_Click"></Button>
        </StackPanel>
    </Grid>
</UserControl>

```

Au niveau de la gestion de l'événement, dans le code attaché au fichier, nous avons ceci lorsque nous voulons contacter le Web service *ASP.NET* :

```

private void CallServiceButton_Click(object sender, RoutedEventArgs e)
{
    // using ASMX Web service
    SimpleASMX.SimpleAsmxSoapClient proxy = new
    ➤ WebServicesApp.SimpleASMX.SimpleAsmxSoapClient();
    proxy.SayHelloToMeCompleted += new
    ➤ EventHandler<WebServicesApp.SimpleASMX
    ➤ .SayHelloToMeCompletedEventArgs>(proxy_SayHelloToMeCompleted);
    proxy.SayHelloToMeAsync(InputText.Text);
}

```

Avec comme méthode pour afficher le résultat :

```
void proxy_SayHelloToMeCompleted(object sender,
➤ WebServicesApp.SimpleASMX.SayHelloToMeCompletedEventArgs e)
{
    OutputText.Text = e.Result;
}
```

Si nous souhaitons contacter le service WCF avec la méthode simple, nous aurons :

```
private void CallServiceButton_Click(object sender, RoutedEventArgs e)
{
    // using WCF Web service with simple type
    SimpleSVC.SimpleWCFClient proxy = new
➤ WebServicesApp.SimpleSVC.SimpleWCFClient();
    proxy.SayHelloToMeCompleted += new
➤ EventHandler<WebServicesApp.SimpleSVC.
➤ SayHelloToMeCompletedEventArgs>(proxy_SayHelloToMeCompleted);
    proxy.SayHelloToMeAsync(InputText.Text);
}
```

Avec exactement la même méthode indiquée au-dessus (proxy_SayHelloToMeCompleted) mais utilisant la définition de WCF :

```
void proxy_SayHelloToMeCompleted
(object sender,
WebServicesApp.SimpleSVC.SayHelloToMeCompletedEventArgs e)
{
    OutputText.Text = e.Result;
}
```

Pour contacter la méthode qui renvoie des données complexes, nous aurons :

```
private void CallServiceButton_Click(object sender, RoutedEventArgs e)
{
    // using WCF Web service with complex type
    SimpleSVC.SimpleWCFClient proxy = new
➤ WebServicesApp.SimpleSVC.SimpleWCFClient();
    proxy.GetPeopleCompleted += new EventHandler<WebServicesApp.
➤ SimpleSVC.GetPeopleCompletedEventArgs>(proxy_GetPeopleCompleted);
    proxy.GetPeopleAsync();
}
```

Avec comme méthode de traitement :

```
void proxy_GetPeopleCompleted(object sender,
    ➤ WebServicesApp.SimpleSVC.GetPeopleCompletedEventArgs e)
{
    // result is Person[]
    SimpleSVC.Person[] people = e.Result;

    PeopleList.Items.Clear();
    PeopleList.ItemsSource = people;
}
```

Ou, si vous voulez utiliser LINQ pour filtrer les données, par exemple rechercher toutes les personnes dont le nom commence par B :

```
void proxy_GetPeopleCompleted(object sender,
    ➤ WebServicesApp.SimpleSVC.GetPeopleCompletedEventArgs e)
{
    // result is Person[]
    SimpleSVC.Person[] people = e.Result;

    var filtered = from ppl in people
                   where ppl.LastName.StartsWith("B")
                   select ppl;

    PeopleList.Items.Clear();
    PeopleList.ItemsSource = people;
}
```

Au niveau du fichier de configuration de Silverlight, nous avons ceci (obtenu lorsque nous avons ajouté la référence au Web service de notre projet *ASP.NET* :

```
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_ISimpleWCF"
          maxBufferSize="65536"
          maxReceivedMessageSize="65536">
          <security mode="None" />
        </binding>
        <binding name="SimpleAsmxSoap" maxBufferSize="65536"
          maxReceivedMessageSize="65536">
          <security mode="None" />
        </binding>
      </basicHttpBinding>
    </bindings>
```

```

<client>
  <endpoint address="http://localhost:1597/WebServicesApp_Web/
    ➤ SimpleWCF.svc"
    binding="basicHttpBinding"
    bindingConfiguration="BasicHttpBinding_ISimpleWCF"
    contract="WebServicesApp.SimpleSVC.ISimpleWCF"
    name="BasicHttpBinding_ISimpleWCF" />
  <endpoint address="http://localhost:1597/WebServicesApp_Web/
    ➤ SimpleAsmx.asmx"
    binding="basicHttpBinding"
    bindingConfiguration="SimpleAsmxSoap"
    contract="WebServicesApp.SimpleASMX.SimpleAsmxSoap"
    name="SimpleAsmxSoap" />
</client>
</system.serviceModel>
</configuration>

```

Au niveau de notre page de test dans le projet *ASP.NET*, nous avons le code suivant :

```

<%@ Page Language="C#" AutoEventWireup="true" %>

<%@ Register Assembly="System.Web.Silverlight"
Namespace="System.Web.UI.SilverlightControls"
TagPrefix="asp" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%;">
<head runat="server">
  <title>Test Page For WebServicesApp</title>
</head>
<body style="height:100%;margin:0;">
  <form id="form1" runat="server" style="height:100%;">
    <asp:ScriptManager ID="ScriptManager1"
      runat="server"></asp:ScriptManager>
    <div style="height:100%;">
      <asp:Silverlight ID="Xaml1" runat="server"
        Source="~/ClientBin/WebServicesApp.xap" Version="2.0"
        Width="100%" Height="100%" />
    </div>
  </form>
</body>
</html>

```

Rendez-vous au chapitre 4, Silverlight et ASP.NET, pour comprendre comment fonctionne ce contrôle ASP.NET Silverlight.

L'application est terminée. WCF est une technologie très vaste. Il existe plusieurs livres sur ce sujet. Si vous souhaitez plus d'informations, n'hésitez pas à utiliser Google ou Live pour trouver tout ce que vous voulez savoir sur cette brique du *Framework .NET*.

3.6 ADO.NET/Silverlight

Silverlight peut exploiter directement les données sans passer par un Web service. Pour cela, nous utiliserons les méthodes présentes dans `System.Data.Services.Client`.

Les exemples de cette section explorent l'association entre des éléments `Order` et `Order_Detail` de la base de données *Northwind* disponible à partir du Centre de téléchargement de Microsoft à cette adresse : <http://go.microsoft.com/fwlink/?linkid=24758>.

Code de l'application

Ce chapitre ne montrera pas de code complet mais uniquement des parties de code. Si vous voulez voir la totalité du code, rendez-vous à la fin de cette section. Vous devez garder en tête ce qui a été vu aux chapitres précédents. Veillez aussi à connaître l'élément XAML `Grid` qui nous permettra d'afficher les données.

Notre application contiendra 5 zones.

La première zone est le `Grid` en question. Il permettra d'afficher les données que nous allons récolter :

```
<Grid.RowDefinitions>
  <RowDefinition Height="40"/>
  <RowDefinition Height="25"/>
  <RowDefinition Height="250"/>
  <RowDefinition Height="25"/>
  <RowDefinition Height="250"/>
</Grid.RowDefinitions>
```

Le deuxième élément constituant notre interface est un `StackPanel` contenant deux boutons :

```
<StackPanel Orientation="Horizontal" Grid.Row="0">
  <Button Content="Get Data"
    Width="75"
    Height="20"
    Margin="10"
```

```

        Click="OnGetCustomers"/>
<Button Content="Get Orders"
        Width="75"
        Height="20"
        Margin="10"
        Click="OnGetOrders"/>
</StackPanel>

```

Ensuite, nous avons besoin de deux `TextBlock` : un pour afficher le client concerné et l'autre pour afficher d'éventuelles erreurs :

```

<TextBlock x:Name="CustomerBlock" Grid.Row="1"
        Text="Customers:" Grid.Column="0" FontFamily="Calibri"/>

<TextBlock x:Name="textBlock" Grid.Row="3"
        Grid.Column="0" FontFamily="Calibri"/>

```

Il nous faudra ensuite deux `DataGrid` pour afficher les clients et les commandes associées aux clients :

```

<data:DataGrid
        Name="dataGridCustomers"
        Grid.Row="2"
        Margin="20"
        AutoGenerateColumns="True"
        ItemsSource="{Binding}"
        >
</data:DataGrid>

<data:DataGrid
        Name="dataGridOrders"
        Grid.Row="4"
        Margin="20"
        AutoGenerateColumns="True"
        ItemsSource="{Binding}"
        >
</data:DataGrid>

```

L'interface utilisateur *Silverlight* complète est définie dans l'exemple XAML suivant :

```

<UserControl x:Class="SilverlightClientApp4.Page"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:data="clr-namespace:System.Windows.Controls;assembly=System
        ➤ .Windows.Controls.Data"
        Width="800" Height="600">

```



```

<Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="25"/>
        <RowDefinition Height="250"/>
        <RowDefinition Height="25"/>
        <RowDefinition Height="250"/>
    </Grid.RowDefinitions>

    <StackPanel Orientation="Horizontal" Grid.Row="0">
        <Button Content="Get Data"
            Width="75"
            Height="20"
            Margin="10"
            Click="OnGetCustomers"/>
        <Button Content="Get Orders"
            Width="75"
            Height="20"
            Margin="10"
            Click="OnGetOrders"/>
    </StackPanel>

    <TextBlock x:Name="CustomerBlock" Grid.Row="1"
        Text="Customers:" Grid.Column="0" FontFamily="Calibri"/>

    <data:DataGrid
        Name="dataGridCustomers"
        Grid.Row="2"
        Margin="20"
        AutoGenerateColumns="True"
        ItemsSource="{Binding}"
    >
    </data:DataGrid>

    <TextBlock x:Name="textBlock" Grid.Row="3"
        Grid.Column="0" FontFamily="Calibri"/>

    <data:DataGrid
        Name="dataGridOrders"
        Grid.Row="4"
        Margin="20"
        AutoGenerateColumns="True"
        ItemsSource="{Binding}"
    >
    </data:DataGrid>
</Grid>
</UserControl>

```

Du côté du code, nous avons 6 actions à effectuer. D'abord il faut créer l'instance de la classe *DataServiceContext* de type *NorthwindEntities* :

```
NorthwindEntities svcContext;
void OnLoaded(object sender, EventArgs args)
{
    svcContext = new NorthwindEntities(
        new Uri("Northwind.svc", UriKind.Relative));

    dataGridCustomers.AutoGeneratingColumn +=
        dataGridCustomers_AutoGeneratingColumn;
    obsvCollCustomers = new ObservableCollection<Customer>();

    OnGetCustomers(null, null);
}
```

Dans la méthode *OnGetCustomers*, nous allons rechercher tous les clients dont le pays est *USA* :

```
void OnGetCustomers(object sender, EventArgs args)
{
    DataServiceQuery<Customer> query =
        (DataServiceQuery<Customer>)
        (from customer in svcContext.Customers
         where customer.Country == "USA"
         select customer);

    try
    {
        query.BeginExecute(GetCustomersCallback, query);
    }
    catch (DataServiceRequestException ex)
    {
        textBlock.Text = "OnGetCustomers Error: " +
            ➡ ex.Response.ToString();
    }
    dataGridOrders.DataContext = null;
}
```

Nous avons utilisé ici une méthode de *CallBack*. Cette méthode s'exécute lorsque le traitement qu'on a défini au même moment est terminé. On obtient le résultat en paramètre de cette fonction *CallBack* :

```
void GetCustomersCallback(IAsyncResult result)
{
```

```

try
{
    DataServiceQuery<Customer> queryResult =
        (DataServiceQuery<Customer>)result.AsyncState;

    IEnumerable<Customer> ienumResults =
        queryResult.EndExecute(result);
    obsvCollCustomers = new ObservableCollection<Customer>();

    foreach (var item in ienumResults)
    {
        obsvCollCustomers.Add(item);
    }

    Dispatcher.BeginInvoke(() =>
    {
        dataGridCustomers.DataContext = obsvCollCustomers;
    });
}
catch (DataServiceRequestException ex)
{
    {
        textBlock.Text = "GetCustomersCallback Error: " +
            ➡ ex.Response.ToString();
    }
}
}

```

Nous avons défini un événement sur la gestion des colonnes. Nous supprimerons deux colonnes qui n'ont pas lieu d'être dans ce Grid-là. Pour ce faire, nous devons annuler la génération si nous rencontrons ces deux colonnes :

```

void dataGridCustomers_AutoGeneratingColumn(object sender,
    DataGridAutoGeneratingColumnEventArgs e)
{
    if (e.PropertyName == "CustomerDemographics" ||
        e.PropertyName == "Orders")
    {
        e.Cancel = true;
    }
}

```

Il faut ensuite gérer la liste des commandes d'un client. Pour cela, nous avons créé un bouton. Sur le clic du bouton **Get Order**, nous allons appeler la méthode `OnGetOrders` :

```

ObservableCollection<Order> obsvCollCustomerOrders;

```

```

void OnGetOrders(object sender, EventArgs args)
{
    Customer selectedCustomer =
        ➡ (Customer)dataGridCustomers.SelectedItem;

    try
    {
        svcContext.BeginLoadProperty(selectedCustomer, "Orders",
                                     OnPropertyLoading, selectedCustomer);
    }
    catch (DataServiceRequestException ex)
    {
        textBlock.Text = "OnGetOrders Error: " +
            ➡ ex.Response.ToString();
    }
}

```

Nous demandons ici à notre *DataContext* les commandes du client actuellement sélectionné :

```

svcContext.BeginLoadProperty(selectedCustomer, "Orders",
                             OnPropertyLoading, selectedCustomer);

```

De la même manière, lorsque le résultat aura été rapatrié, la méthode *OnPropertyLoading* sera appelée et cette fonction traitera les différentes informations, comme l'ajout dans le *DataGrid* :

```

void OnPropertyLoading(IAsyncResult result)
{
    void OnPropertyLoading(IAsyncResult result)
    {
        Customer resultCustomer = result.AsyncState as Customer;

        try
        {
            svcContext.EndLoadProperty(result);

            if (resultCustomer.Orders.Count == 0)
            {
                textBlock.Text = "Customer has no orders";
                dataGridOrders.DataContext = null;
                return;
            }
            else
                textBlock.Text = "Customer orders:";
        }
    }
}

```

```

        obsvCollCustomerOrders =
            new ObservableCollection<Order>();
        foreach (Order order in resultCustomer.Orders)
        {
            obsvCollCustomerOrders.Add(order);
        }

        Dispatcher.BeginInvoke(() =>
        {
            dataGridOrders.DataContext = obsvCollCustomerOrders;
        });
    }
    catch (DataServiceRequestException ex)
    {
        textBlock.Text = "OnPropertyLoading Error: " +
            ➡ ex.Response.ToString();
    }
}

```

Voici le code de notre *Page.xaml.cs* :

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

using SilverlightClientApp4.NorthwindSvc;
using System.Data.Services.Client;
using System.Collections.ObjectModel;

namespace SilverlightClientApp4
{
    public partial class Page : UserControl
    {
        NorthwindEntities svcContext;
        ObservableCollection<Customer> obsvCollCustomers;
        ObservableCollection<Order> obsvCollCustomerOrders;

        public Page()
        {

```

```

        InitializeComponent();
        this.Loaded += OnLoaded;

    }

    void OnLoaded(object sender, EventArgs args)
    {
        svcContext = new NorthwindEntities(
            new Uri("Northwind.svc", UriKind.Relative));

        dataGridCustomers.AutoGeneratingColumn +=
            dataGridCustomers_AutoGeneratingColumn;
        obsvCollCustomers = new ObservableCollection<Customer>();

        OnGetCustomers(null, null);
    }

    void OnGetOrders(object sender, EventArgs args)
    {
        Customer selectedCustomer =
            ➡ (Customer)dataGridCustomers.SelectedItem;

        try
        {
            svcContext.BeginLoadProperty(selectedCustomer,
                "Orders",
                OnPropertyLoading, selectedCustomer);
        }
        catch (DataServiceRequestException ex)
        {
            {
                textBlock.Text = "OnGetOrders Error: " +
                    ➡ ex.Response.ToString();
            }
        }
    }

    void OnPropertyLoading(IAsyncResult result)
    {
        Customer resultCustomer = result.AsyncState as Customer;

        try
        {
            svcContext.EndLoadProperty(result);

            if (resultCustomer.Orders.Count == 0)
            {
                textBlock.Text = "Customer has no orders";
            }
        }
    }

```

```

        dataGridOrders.DataContext = null;
        return;
    }
    else
        textBlock.Text = "Customer orders:";

    obsvCollCustomerOrders =
        new ObservableCollection<Order>();
    foreach (Order order in resultCustomer.Orders)
    {
        obsvCollCustomerOrders.Add(order);
    }

    Dispatcher.BeginInvoke(() =>
    {
        dataGridOrders.DataContext = obsvCollCustomerOrders;
    });
}
catch (DataServiceRequestException ex)
{
    textBlock.Text = "OnPropertyLoading Error: " +
        ➡ ex.Response.ToString();
}
}

void OnGetCustomers(object sender, EventArgs args)
{
    DataServiceQuery<Customer> query =
        (DataServiceQuery<Customer>)
        (from customer in svcContext.Customers
         where customer.Country == "USA"
         select customer);

    try
    {
        query.BeginExecute(GetCustomersCallback, query);
    }
    catch (DataServiceRequestException ex)
    {
        textBlock.Text = "OnGetCustomers Error: " +
            ➡ ex.Response.ToString();
    }
    dataGridOrders.DataContext = null;
}

void GetCustomersCallback(IAsyncResult result)
{

```

```

try
{
    DataServiceQuery<Customer> queryResult =
        (DataServiceQuery<Customer>) result.AsyncState;

    IEnumerable<Customer> ienumResults =
        queryResult.EndExecute(result);
    obsvCollCustomers = new
    ➤ ObservableCollection<Customer>();

    foreach (var item in ienumResults)
    {
        obsvCollCustomers.Add(item);
    }

    Dispatcher.BeginInvoke(() =>
    {
        dataGridCustomers.DataContext = obsvCollCustomers;
    });
}
catch (DataServiceRequestException ex)
{
    {
        textBlock.Text = "GetCustomersCallback Error: " +
        ➤ ex.Response.ToString();
    }
}

void dataGridCustomers_AutoGeneratingColumn(object sender,
    DataGridAutoGeneratingColumnEventArgs e)
{
    if (e.PropertyName == "CustomerDemographics" ||
        e.PropertyName == "Orders")
    {
        e.Cancel = true;
    }
}
}

```


3.7 Créez un widget météo

Avec tout ce que nous avons vu, vous êtes maintenant capable de comprendre comment créer un *Widget* indiquant la météo.



► Fig. 3.2 :
Widget météo

De nombreux sites Internet diffusent des informations de météo via Web services. Si vous avez un *iPhone* et que vous avez déjà manipulé l'application météo, vous avez alors déjà utilisé le Web service météo de *Yahoo*. En effet, l'application a besoin d'une connexion data pour aller rechercher l'information sur *Yahoo* et vous l'afficher ensuite.

Nous utiliserons un autre Web service. Vous pourrez trouver un grand nombre de services sur le site suivant : <http://a4472706772.api.wxbug.net/>.

La première chose à faire est d'ajouter la référence au Web service dans votre projet Silverlight. Ceci crée un fichier *ServiceReferences.ClientConfig* :

```
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="WeatherBugWebServicesSoap"
          maxBufferSize="2147483647"
          maxReceivedMessageSize="2147483647">
          <security mode="None" />
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://a4472706772.api.wxbug.net/
        ➡ weatherservice.asmx"
        binding="basicHttpBinding"
        bindingConfiguration="WeatherBugWebServicesSoap"
        contract="WeatherService.WeatherBugWebServicesSoap"
        name="WeatherBugWebServicesSoap" />
    </client>
  </system.serviceModel>
</configuration>
```

i Changement au niveau Web service

Le code présent ici ne l'est qu'à titre d'exemples. Il se peut que le Web service change de point d'entrée avant que ce livre sorte. Veuillez donc vous rendre sur l'URL indiquée précédemment pour avoir accès au point d'accès correct.

Pour notre interface, nous allons utiliser un grand nombre d'images. Vous pourrez retrouver tout ce code et les images sur le site de Micro Application :

```
<UserControl.Resources>
  <ImageBrush ImageSource="Resources/BLUE-base.png" x:Key="blueBase" />
  <ImageBrush ImageSource="Resources/GRAY-base.png" x:Key="grayBase" />
  <ImageBrush ImageSource="Resources/divider-vertical.png"
x:Key="dividerVertical" Opacity="0.6" />
  <ImageBrush ImageSource="Resources/divider-horizontal.png"
x:Key="dividerHorizontal" Opacity="0.6" />
  <BitmapImage UriSource="Resources/BLACK-highlight-01.png"
x:Key="blackHighlight" />
  <BitmapImage UriSource="Resources/undocked_cloudy.png"
x:Key="CloudyBig" />
  <BitmapImage UriSource="Resources/26.png" x:Key="Cloudy" />
  <BitmapImage UriSource="Resources/undocked_few-showers.png"
x:Key="FewShowersBig" />
  <BitmapImage UriSource="Resources/5.png" x:Key="FewShowers" />
  <BitmapImage UriSource="Resources/undocked_foggy.png"
x:Key="FoggyBig" />
  <BitmapImage UriSource="Resources/19.png" x:Key="Foggy" />
  <BitmapImage UriSource="Resources/undocked_hail.png" x:Key="HailBig" />
  <BitmapImage UriSource="Resources/17.png" x:Key="Hail" />
  <BitmapImage UriSource="Resources/undocked_partly-cloudy.png"
x:Key="PartlyCloudyBig" />
  <BitmapImage UriSource="Resources/29.png" x:Key="PartlyCloudy" />
  <BitmapImage UriSource="Resources/undocked_rainy.png"
x:Key="RainyBig" />
  <BitmapImage UriSource="Resources/8.png" x:Key="Rainy" />
  <BitmapImage UriSource="Resources/undocked_snow.png" x:Key="SnowBig" />
  <BitmapImage UriSource="Resources/13.png" x:Key="Snow" />
  <BitmapImage UriSource="Resources/undocked_sun.png" x:Key="SunBig" />
  <BitmapImage UriSource="Resources/undocked_moon-full.png"
x:Key="MoonBig" />
  <BitmapImage UriSource="Resources/31.png" x:Key="Sun" />
  <BitmapImage UriSource="Resources/undocked_thunderstorm.png"
x:Key="ThunderstormBig" />
  <BitmapImage UriSource="Resources/1.png" x:Key="Thunderstorm" />
  <BitmapImage UriSource="Resources/undocked_windy.png"
x:Key="WindyBig" />
  <BitmapImage UriSource="Resources/23.png" x:Key="Windy" />
```

```
<SolidColorBrush Color="White" Opacity="0.5"
  x:Key="zipCodeBackground" />
</UserControl.Resources>
```

Nous utiliserons ces ressources de la manière suivante :

```
<Canvas Width="71" Background="{StaticResource dividerVertical}"
  ➡ Margin="5, 0, 0, 0">
```

Pour plus d'informations sur le binding, rendez-vous au chapitre Le langage XAML, où l'on explique son fonctionnement exact.

Voici le XAML complet de l'application qui utilise les ressources :

```
<UserControl x:Class="WeatherWidget.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="320"
  Height="240"
  >
  <UserControl.Resources>
    <ImageBrush ImageSource="Resources/BLUE-base.png"
      x:Key="blueBase" />
    <ImageBrush ImageSource="Resources/GRAY-base.png"
      x:Key="grayBase" />
    <ImageBrush ImageSource="Resources/divider-vertical.png"
      x:Key="dividerVertical" Opacity="0.6" />
    <ImageBrush ImageSource="Resources/divider-horizontal.png"
      x:Key="dividerHorizontal" Opacity="0.6" />
    <BitmapImage UriSource="Resources/BLACK-highlight-01.png"
      x:Key="blackHighlight" />
    <BitmapImage UriSource="Resources/undocked_cloudy.png"
      x:Key="CloudyBig" />
    <BitmapImage UriSource="Resources/26.png" x:Key="Cloudy" />
    <BitmapImage UriSource="Resources/undocked_few-showers.png"
      x:Key="FewShowersBig" />
    <BitmapImage UriSource="Resources/5.png" x:Key="FewShowers" />
    <BitmapImage UriSource="Resources/undocked_foggy.png"
      x:Key="FoggyBig" />
    <BitmapImage UriSource="Resources/19.png" x:Key="Foggy" />
    <BitmapImage UriSource="Resources/undocked_hail.png"
      x:Key="HailBig" />
    <BitmapImage UriSource="Resources/17.png" x:Key="Hail" />
    <BitmapImage UriSource="Resources/undocked_partly-cloudy.png"
      x:Key="PartlyCloudyBig" />
    <BitmapImage UriSource="Resources/29.png" x:Key="PartlyCloudy" />
    <BitmapImage UriSource="Resources/undocked_rainy.png"
```

```

        x:Key="RainyBig" />
<BitmapImage UriSource="Resources/8.png" x:Key="Rainy" />
<BitmapImage UriSource="Resources/undocked_snow.png"
    x:Key="SnowBig" />
<BitmapImage UriSource="Resources/13.png" x:Key="Snow" />
<BitmapImage UriSource="Resources/undocked_sun.png"
    x:Key="SunBig" />
<BitmapImage UriSource="Resources/undocked_moon-full.png"
    x:Key="MoonBig" />
<BitmapImage UriSource="Resources/31.png" x:Key="Sun" />
<BitmapImage UriSource="Resources/undocked_thunderstorm.png"
    x:Key="ThunderstormBig" />
<BitmapImage UriSource="Resources/1.png" x:Key="Thunderstorm" />
<BitmapImage UriSource="Resources/undocked_windy.png"
    x:Key="WindyBig" />
<BitmapImage UriSource="Resources/23.png" x:Key="Windy" />
<SolidColorBrush Color="White" Opacity="0.5"
    x:Key="zipCodeBackground" />
</UserControl.Resources>
<Canvas x:Name="LayoutRoot" Width="264" Height="194"
    Background="{StaticResource blueBase}">
    <Canvas Canvas.Top="13" Canvas.Left="13" Width="230"
        Height="160">
        <StackPanel x:Name="ConditionsScreen" >
            <Canvas Height="90">
                <Image x:Name="ConditionsOverlay"
                    Margin="-13,-13,0,0"
                    Source="{StaticResource SunBig}" />
            <StackPanel>
                <TextBlock Width="225" Height="37"
                    TextAlignment="Right"
                    FontSize="34" x:Name="TodayTemp" />
                <TextBlock Width="225" Height="14"
                    TextAlignment="Right"
                    x:Name="TodayDescription" />
                <TextBlock Width="225" Height="14"
                    TextAlignment="Right"
                    x:Name="TodayRange" />
                <TextBlock Width="225" Height="14"
                    TextAlignment="Right" x:Name="City"
                    Text="Fetching data..." />
            </StackPanel>
        </Canvas>
        <StackPanel Orientation="Horizontal" Height="53">
            <Canvas Width="71"
                Background="{StaticResource
                    ➡ dividerVertical}"

```

```

        Margin="5, 0, 0, 0">
        <TextBlock FontSize="11" x:Name="TomorrowName"
            Foreground="White" Opacity="0.5" />
        <Image x:Name="TomorrowImage" Canvas.Top="17"
            Canvas.Left="23" />
        <TextBlock Canvas.Top="20" x:Name="TomorrowHi"
            Foreground="White" />
        <TextBlock Canvas.Top="35" x:Name="TomorrowLo"
            Foreground="White" Opacity="0.5" />
    </Canvas>
</Canvas Width="71"
    Background="{StaticResource dividerVertical}"
    Margin="5, 0, 0, 0">
    <TextBlock FontSize="11" x:Name="DayAfterName"
        Foreground="White" Opacity="0.5" />
    <Image x:Name="DayAfterImage" Canvas.Top="17"
        Canvas.Left="23" />
    <TextBlock Canvas.Top="20" x:Name="DayAfterHi"
        Foreground="White" />
    <TextBlock Canvas.Top="35" x:Name="DayAfterLo"
        Foreground="White" Opacity="0.5" />
</Canvas>
<Canvas Width="71" Margin="5, 0, 0, 0">
    <TextBlock FontSize="11" x:Name="TwoDaysAwayName"
        Foreground="White" Opacity="0.5" />
    <Image x:Name="TwoDaysAwayImage" Canvas.Top="17"
        Canvas.Left="23" />
    <TextBlock Canvas.Top="20" x:Name="TwoDaysAwayHi"
        Foreground="White" />
    <TextBlock Canvas.Top="35" x:Name="TwoDaysAwayLo"
        Foreground="White" Opacity="0.5" />
</Canvas>
</StackPanel>
<StackPanel Height="17"
    Background="{StaticResource dividerHorizontal}"
    Orientation="Horizontal" >
    <TextBlock Text="Refresh data" Margin="5, 0, 5, 0"
        Width="105" Foreground="White" Opacity="0.5"
        MouseLeftButtonUp="TextBlock_MouseLeft
        ➡ ButtonUp_Refresh" />
    <TextBlock Text="Change ZIP" Margin="5, 0, 5, 0"
        Width="105" TextAlignment="Right"
        Foreground="White"
        Opacity="0.5"
        MouseLeftButtonUp="TextBlock_MouseLeft
        ButtonUp_Zip" />
</StackPanel>

```

```

        </StackPanel>
        <StackPanel x:Name="ZipCodeScreen"
            HorizontalAlignment="Center"
            Visibility="Collapsed" Width="230">
            <TextBlock Margin="0, 50, 0, 0"
                Text="Please enter ZIP code:"
                Width="130" />
            <StackPanel Orientation="Horizontal" Width="120" >
                <TextBox x:Name="ZipCode" Width="55" MaxLength="5"
                    FontSize="14"
                    Background="{StaticResource
                        zipCodeBackground}" />
                <Button Content="Change"
                    Click="Button_Click" Margin="10, 0, 0, 0" />
            </StackPanel>
            <TextBlock Margin="0, 20, 0, 0"
                Width="220"
                Text="Weather data courtesy of WeatherBug."
                />
        </StackPanel>
        <Image IsHitTestVisible="False"
            Source="{StaticResource blackHighlight}" />
    </Canvas>
</Canvas>
</UserControl>

```

Dans le code C# attaché à ce fichier XAML, nous avons tout ce qui est connexion au Web service. C'est là que nous irons rechercher les données concernant la météo.

Nous avons une classe Page qui dérive de UserControl, comme dans chaque projet Silverlight :

```

using System;
using System.ServiceModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using WeatherWidget.WeatherService;
using System.Windows.Browser;

namespace WeatherWidget
{
    public enum WeatherConditions
    {
        Cloudy,

```

```

        FewShowers,
        Foggy,
        Hail,
        PartlyCloudy,
        Rainy,
        Snow,
        Sun,
        Moon,
        Thunderstorm,
        Windy
    }

    public partial class Page : UserControl
    {
        public Page()
        {
        }

    }
}

```

Nous avons une énumération qui permet de choisir l'image qui sera affichée. Dans les ressources, nous avons déclaré deux arrière-plans. Un pour les bonnes journées et l'autre pour les mauvaises journées. Par défaut, le soleil est affiché :

```

<Canvas x:Name="LayoutRoot" Width="264" Height="194"
        Background="{StaticResource blueBase}">

```

Au niveau des ressources, nous avons bien :

```

<ImageBrush ImageSource="Resources/BLUE-base.png"
            x:Key="blueBase" />
<ImageBrush ImageSource="Resources/GRAY-base.png"
            x:Key="grayBase" />

```

Grâce à l'ajout de la référence vers le Web service, nous pouvons créer un projet de type : WeatherBugWebServicesSoapClient

```

private WeatherBugWebServicesSoapClient proxy;

```

En plus de cet objet, nous avons besoin de deux variables :

```

// Veuillez obtenir une clef pour l'api à cette adresse :
// http://www.weatherbug.com/api/default.asp
private const string apiCode = "";
private int zipCode = 98101;

```

i Le code postal en dur

Nous avons ici imposé un code postal à l'application. Nous aurions pu décider que ce code serait donné par l'utilisateur. Il s'agit seulement d'un code de base. L'utilisateur est apte à changer ce code via une *textbox*. On pourra alors faire une nouvelle fois appel au Web service avec une nouvelle valeur pour le code postal.

Dans le constructeur de notre application, nous allons créer et initialiser ce qui concerne notre Web service :

```
public Page()
{
    // Required to initialize variables
    InitializeComponent();

    // Check that user provided an API key
    if (!String.IsNullOrEmpty(apiCode))
    {
        // Initialize Web service
        proxy = new WeatherBugWebServicesSoapClient();

        // Create event handlers for service methods
        proxy.GetForecastByUSZipCodeCompleted +=
            new EventHandler<GetForecastByUSZipCodeCompletedEventArgs>
                (proxy_GetForecastByUSZipCodeCompleted);
        proxy.GetLiveWeatherByUSZipCodeCompleted +=
            new EventHandler<GetLiveWeatherByUSZipCodeCompletedEventArgs>
                (proxy_GetLiveWeatherByUSZipCodeCompleted);

        // Update display
        UpdateDisplay();
    }
    else
    {
        // If no API key was provided, alert the user and disable the UI
        City.Text = "No API key provided";
        ConditionsScreen.IsHitTestVisible = false;
        ZipCodeScreen.IsHitTestVisible = false;
    }
}
```

La première fonction utilisée est `InitializeComponent`. Elle est présente dans toutes les applications. Elle permet d'initialiser les différents composants *XAML* de l'application.

Par exemple, dans notre cas :

```
[System.Diagnostics.DebuggerNonUserCodeAttribute()]
public void InitializeComponent() {
    if (_contentLoaded) {
        return;
    }
    _contentLoaded = true;
    System.Windows.Application.LoadComponent(this, new System.Uri
    ➤ ("/WeatherWidget;component/Page.xaml", System.UriKind.Relative));
    this.LayoutRoot = ((System.Windows.Controls.Canvas) (this.FindName
    ➤ ("LayoutRoot")));
    this.ConditionsScreen = ((System.Windows.Controls.StackPanel)
    ➤ (this.FindName("ConditionsScreen")));
    this.ConditionsOverlay = ((System.Windows.Controls.Image)
    ➤ (this.FindName("ConditionsOverlay")));
    this.TodayTemp = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TodayTemp")));
    this.TodayDescription =
    ➤ ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TodayDescription")));
    this.TodayRange = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TodayRange")));
    this.City = ((System.Windows.Controls.TextBlock)
    ➤ (this.FindName("City")));
    this.TomorrowName = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TomorrowName")));
    this.TomorrowImage = ((System.Windows.Controls.Image) (this.FindName
    ➤ ("TomorrowImage")));
    this.TomorrowHi = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TomorrowHi")));
    this.TomorrowLo = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TomorrowLo")));
    this.DayAfterName = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("DayAfterName")));
    this.DayAfterImage = ((System.Windows.Controls.Image) (this.FindName
    ➤ ("DayAfterImage")));
    this.DayAfterHi = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("DayAfterHi")));
    this.DayAfterLo = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("DayAfterLo")));
    this.TwoDaysAwayName = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TwoDaysAwayName")));
    this.TwoDaysAwayImage =
    ➤ ((System.Windows.Controls.Image) (this.FindName
    ➤ ("TwoDaysAwayImage")));
    this.TwoDaysAwayHi = ((System.Windows.Controls.TextBlock)
    ➤ (this.FindName("TwoDaysAwayHi")));
    this.TwoDaysAwayLo = ((System.Windows.Controls.TextBlock) (this.FindName
    ➤ ("TwoDaysAwayLo")));
```

```

    this.ZipCodeScreen = ((System.Windows.Controls.StackPanel)
➡ (this.FindName("ZipCodeScreen")));
    this.ZipCode = ((System.Windows.Controls.TextBox)
➡ (this.FindName("ZipCode")));
}

```

Ensuite, nous créons notre instance du type objet `WeatherBugWebServiceSoapClient` :

```
proxy = new WeatherBugWebServicesSoapClient();
```

Nous avons déclaré ce proxy précédemment dans le code afin d'avoir accès à cette variable partout dans le reste des fonctions. C'est donc un variable globale.

Nous ajoutons après deux événements qui seront interceptés par les deux fonctions suivantes :

- `proxy_GetForecastByUSZipCodeCompleted` ;
- `proxy_GetLiveWeatherByUSZipCodeCompleted`.

Le type des arguments est défini dans la référence. Nous avons par exemple `GetLiveWeatherByUSZipCodeCompletedEventArgs` :

```

[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel",
➡ "3.0.0.0")]
public partial class GetLiveWeatherByUSZipCodeCompletedEventArgs :
    System.ComponentModel.AsyncCompletedEventArgs {

    private object[] results;

    public GetLiveWeatherByUSZipCodeCompletedEventArgs
        (object[] results, System.Exception exception,
         bool cancelled, object userState) :
        base(exception, cancelled, userState) {
        this.results = results;
    }

    public WeatherWidget.WeatherService.LiveWeatherData Result {
        get {
            base.RaiseExceptionIfNecessary();
            return ((WeatherWidget.WeatherService.LiveWeatherData)
                (this.results[0]));
        }
    }
}

```

Ce code est automatiquement généré par l'ajout de la référence au Web service. Il est toutefois important de savoir comment cela fonctionne. Vous serez parfois dans l'obligation de déclarer vous-même ces objets. Quoi qu'il en soit, *Visual Studio* gère pour vous une grande partie du projet.

Les deux fonctions événements doivent être déclarées. La première permet de connaître les prévisions météo et la deuxième le temps actuel :

```
void proxy_GetForecastByUSZipCodeCompleted(object sender,
    GetForecastByUSZipCodeCompletedEventArgs e)
{
    ArrayOfAnyType forecast = e.Result;

    // Set the remaining three days, except for today
    for (int i = 1; i < 4; i++)
    {
        ApiForecastData today = (ApiForecastData)forecast[i];

        setDay(i, today.Title, parseTemp(today.TempHigh),
            parseTemp(today.TempLow), int.Parse(today.ConditionID));
    }

    // Then call for today's live data, and pass the conditions we'll
    ➡ need later
    proxy.GetLiveWeatherByUSZipCodeAsync(zipCode.ToString(),
        UnitType.English, apiCode, forecast[0]);
}

void proxy_GetLiveWeatherByUSZipCodeCompleted(object sender,
    GetLiveWeatherByUSZipCodeCompletedEventArgs e)
{
    LiveWeatherData today = e.Result;
    ApiForecastData todayForecast = (ApiForecastData)e.UserState;

    // Set today's conditions
    setToday(today.City, todayForecast.ShortPrediction,
        float.Parse(today.Temperature),
        ➡ parseTemp(today.TemperatureHigh),
        parseTemp(today.TemperatureLow),
        ➡ int.Parse(todayForecast.ConditionID),
        !todayForecast.IsNight);
}
```

Dans la première fonction, nous avons besoin de `setDay`. Cette fonction va rechercher les éléments *XAML* pour y afficher les bonnes valeurs, au bon endroit. Elle emploie une fonction "utilitaire" pour convertir le temps :

```
static int parseTemp(string input)
{
    int temp;

    // Remove trailing unrecognized characters
    input = input.TrimEnd('\xFFFD');

    // If the hi/lo is unavailable, set it to 0
    if (String.IsNullOrEmpty(input) || input == "--")
    {
        temp = 0;
    }
    else
    {
        temp = int.Parse(input);
    }

    return temp;
}
```

`SetDay` permet de mettre à jour les différents éléments de notre *XAML* :

```
private void setDay(int offset, string name,
    int hi, int lo, int weatherConditions)
{
    TextBlock todayName;
    TextBlock todayHi;
    TextBlock todayLo;
    Image todayImage;

    switch (offset)
    {
        case 1:
            todayName = TomorrowName;
            todayHi = TomorrowHi;
            todayLo = TomorrowLo;
            todayImage = TomorrowImage;
            break;
        case 2:
            todayName = DayAfterName;
            todayHi = DayAfterHi;
            todayLo = DayAfterLo;
            todayImage = DayAfterImage;
    }
}
```

```

        break;
    case 3:
    default:
        todayName = TwoDaysAwayName;
        todayHi = TwoDaysAwayHi;
        todayLo = TwoDaysAwayLo;
        todayImage = TwoDaysAwayImage;
        break;
    }

    todayName.Text = name.ToString();
    todayHi.Text = hi.ToString() + "°";
    todayLo.Text = lo.ToString() + "°";

    todayImage.Source = mapConditionsToImage(mapCodesToConditions
        (weatherConditions, true), false);
}

```

On voit que dans le switch, on recherche les bons éléments XAML au bon endroit. Cela se fera en fonction du choix de l'affichage des prévisions pour le jour un, deux ou trois et de l'*offset* qui est passé en paramètre.

Ensuite, on indique les bonnes valeurs, valeurs passées en paramètre de notre fonction.

On mappe après l'image qui correspond aux valeurs :

```

private BitmapImage mapConditionsToImage(WeatherConditions
    weatherConditions, bool isBig)
{
    // Build the resource name - something like RainyBig
    string resourceName = Enum.GetName(typeof(WeatherConditions),
        weatherConditions) + (isBig ? "Big" : "");
    return (BitmapImage)Resources[resourceName];
}

```

La valeur passée en paramètre est obtenue à l'aide d'une fonction `MapCodesToConditions`. Cette fonction étant peu complexe mais excessivement longue, nous l'avons placée à la fin de cet exemple, dans une partie portant le nom de la fonction.

L'autre fonction dont nous avons besoin est celle qui va afficher le temps qu'il fait actuellement dehors. Cette fonction se nomme `SetToday` et nous l'avons utilisée dans `proxy_GetLiveWeatherByUSZipCodeCompleted`. Elle suit plus ou moins le même principe :

```

private void setToday(string cityName, string description,
    float temp, int hi, int lo, int weatherConditions, bool isDay)

```

```

{
    // Set the weather parameters
    City.Text = cityName;
    TodayTemp.Text = temp.ToString() + "°";
    TodayDescription.Text = description;
    TodayRange.Text = hi + "° - " + lo + "°";

    // Set the correct background according to the conditions
    WeatherConditions currentConditions =
        mapCodesToConditions(weatherConditions, isDay);

    // Set correct main image according to the conditions
    ConditionsOverlay.Source = mapConditionsToImage
        (currentConditions, true);

    if (!isDay ||
        (currentConditions == WeatherConditions.Moon) ||
        (currentConditions == WeatherConditions.Cloudy) ||
        (currentConditions == WeatherConditions.Foggy) ||
        (currentConditions == WeatherConditions.Hail) ||
        (currentConditions == WeatherConditions.Rainy) ||
        (currentConditions == WeatherConditions.Snow) ||
        (currentConditions == WeatherConditions.Thunderstorm))
    {
        LayoutRoot.Background =
            (ImageBrush)Resources["grayBase"];
    }
    else
    {
        LayoutRoot.Background =
            (ImageBrush)Resources["blueBase"];
    }
}

```

On met une valeur à quelques champs et on fait appel aux mêmes méthodes que dans la première fonction : `mapConditionsToImage` et `mapCodesToConditions`.

Le constructeur fait ensuite appel à une fonction `UpdateDisplay` pour mettre à jour toutes les données :

```

private void UpdateDisplay()
{
    // Call for the 4 day forecast first
    proxy.GetForecastByUSZipCodeAsync
        (zipCode.ToString(), UnitType.English, apiCode);
}

```

Cette méthode déclenche les deux autres événements qui mettent à jour les données de notre application Silverlight. Nous l'utiliserons un peu partout dans notre code.

Comme indiqué dans la remarque précédemment, nous disposons d'un bouton qui permet de mettre à jour le code postal. Nous avons donc ajouté un événement sur le clic du bouton :

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    bool error = false;

    if (ZipCode.Text.Length == ZipCode.MaxLength)
    {
        try
        {
            zipCode = int.Parse(ZipCode.Text);
            UpdateDisplay();

            // Switch to conditions screen
            ConditionsScreen.Visibility = Visibility.Visible;
            ZipCodeScreen.Visibility = Visibility.Collapsed;

        }
        catch (FormatException)
        {
            error = true;
        }
    } else {
        error = true;
    }

    if(error)
        ZipCode.Foreground = new SolidColorBrush(Colors.Red);
}
```

Ce bouton va rechercher le code postal dans une *textbox*, le transforme en chiffre et met à jour le contenu de notre application.

Nous avons encore deux événements à gérer du côté de notre code. `TextBlock_MouseLeftButtonUp_Zip` est l'un de ces événements. Il permet en fait d'afficher la *textbox* pour modifier le code postal :

```
private void TextBlock_MouseLeftButtonUp_Zip
(object sender, MouseButtonEventArgs e)
{
    ZipCode.Text = "";
}
```

```

ZipCode.Foreground = new SolidColorBrush
    (Colors.Black);

// Switch to ZIP code selection screen
ConditionsScreen.Visibility = Visibility.Collapsed;
ZipCodeScreen.Visibility = Visibility.Visible;
}

```

Enfin, nous disposons d'un bouton permettant de rafraîchir le contenu. Pour cela, il suffit d'appeler la méthode déjà déclarée précédemment :

```

private void TextBlock_MouseLeftButtonUp_Refresh
    (object sender, MouseButtonEventArgs e)
{
    UpdateDisplay();
}

```

L'application du côté Silverlight est terminée. Reste à créer une page *HTML* de test pour afficher notre application dans un navigateur.

Au niveau de notre page *HTML*, rien de très difficile :

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<!-- saved from url=(0014)about:internet -->
<head>
    <title>Silverlight Project Test Page </title>

    <style type="text/css">
html, body {
    height: 100%;
    overflow: auto;
}
body {
    padding: 0;
    margin: 0;
}
#silverlightControlHost {
    height: 100%;
}
</style>

    <script type="text/javascript">
        function onSilverlightError(sender, args) {

```



```

        var appSource = "";
        if (sender != null && sender != 0) {
            appSource = sender.getHost().Source;
        }
        var errorType = args.ErrorType;
        var iErrorCode = args.ErrorCode;

        var errMsg =
"Unhandled Error in Silverlight 2 Application " +
appSource + "\n" ;

        errMsg += "Code: " + iErrorCode + "      \n";
        errMsg += "Category: " + errorType + "      \n";
        errMsg += "Message: " + args.ErrorMessage + "      \n";

        if (errorType == "ParserError")
        {
            errMsg += "File: " + args.xamlFile + "      \n";
            errMsg += "Line: " + args.lineNumber + "      \n";
            errMsg += "Position: " + args.charPosition + "      \n";
        }
        else if (errorType == "RuntimeError")
        {
            if (args.lineNumber != 0)
            {
                errMsg += "Line: " + args.lineNumber + "      \n";
                errMsg += "Position: " + args.charPosition + "      \n";
            }
            errMsg += "MethodName: " + args.methodName + "      \n";
        }

        throw new Error(errMsg);
    }
</script>
</head>

<body>
    <!-- Runtime errors from Silverlight will be displayed here.
    This will contain debugging information and should be
    removed or hidden when debugging is completed -->
    <div id='errorLocation' style="font-size: small;color: Gray;">
    </div>

    <div id="silverlightControlHost">
        <object data="data:application/x-silverlight-2,"
            type="application/x-silverlight-2" width="100%"
            height="100%">
            <param name="source"

```

```

        value="ClientBin/WeatherWidget.xap"/>
        <param name="onerror"
            value="onSilverlightError" />
        <param name="background"
            value="white" />
        <param name="minRuntimeVersion"
            value="2.0.30923.0" />
        <param name="autoUpgrade" value="true" />
        <a href="http://go.microsoft.com/fwlink/?LinkId=124807"
            style="text-decoration: none;">
            
        </a>
    </object>
    <iframe
        visibility="hidden" height="0" width="0" border="0px"></iframe>
</div>
</body>
</html>

```

Notre premier exemple est terminé. Nous avons vu ici comment contacter un Web service. Dans le prochain exemple, nous apprendrons à manipuler les données RSS.

MapCodesToConditions

```

static WeatherConditions mapCodesToConditions(int weatherConditions,
    bool isDay)
{
    WeatherConditions conditions;

    switch (weatherConditions)
    {
        case 1:
        case 13:
        case 24:
        case 34:
        case 66:
        case 68:
        case 73:
            conditions = WeatherConditions.Cloudy;
            break;
        case 9:
        case 19:
        case 21:
        case 25:

```

```
case 27:
case 28:
case 32:
case 36:
case 46:
case 47:
case 48:
case 49:
case 56:
case 57:
case 60:
case 61:
case 84:
case 85:
case 86:
case 90:
case 91:
case 92:
case 96:
case 97:
case 98:
case 99:
case 100:
case 101:
case 120:
case 121:
case 122:
case 123:
case 124:
case 125:
case 129:
case 130:
case 131:
case 135:
case 136:
case 137:
case 140:
case 142:
case 144:
case 145:
case 152:
case 153:
case 155:
case 157:
    conditions = WeatherConditions.FewShowers;
    break;
case 23:
```

```
case 33:
case 51:
    conditions = WeatherConditions.Foggy;
    break;
case 2:
case 3:
case 16:
case 67:
case 71:
case 72:
    conditions = WeatherConditions.PartlyCloudy;
    break;
case 5:
case 14:
case 15:
case 20:
case 38:
case 41:
case 42:
case 45:
case 52:
case 58:
case 59:
case 63:
case 81:
case 82:
case 83:
case 87:
case 88:
case 89:
case 108:
case 109:
case 110:
case 114:
case 115:
case 116:
case 132:
case 133:
case 134:
case 139:
case 141:
case 148:
case 150:
case 156:
    conditions = WeatherConditions.Rainy;
    break;
case 8:
```

```
case 11:
case 12:
case 29:
case 39:
case 40:
case 43:
case 44:
case 54:
case 55:
case 62:
case 69:
case 74:
case 78:
case 79:
case 80:
case 102:
case 103:
case 104:
case 111:
case 112:
case 113:
case 117:
case 118:
case 119:
case 126:
case 127:
case 128:
case 138:
case 146:
case 149:
case 151:
case 154:
    conditions = WeatherConditions.Snow;
    break;
case 0:
case 7:
case 10:
case 17:
case 26:
case 31:
case 35:
case 37:
case 64:
case 65:
case 70:
case 75:
case 76:
```

```

        case 77:
        default:
            conditions = (isDay) ? WeatherConditions.Sun :
                WeatherConditions.Moon;
            break;
        case 6:
        case 18:
        case 22:
        case 30:
        case 53:
        case 93:
        case 94:
        case 95:
        case 105:
        case 106:
        case 107:
        case 143:
        case 147:
            conditions = WeatherConditions.Thunderstorm;
            break;
        case 50:
            conditions = WeatherConditions.Windy;
            break;
    }

    return conditions;
}

```

3.8 Traitez un flux de données RSS

RSS est un format de données basé sur XML. Il permet la plupart du temps de stocker des actualités (titre, description, date etc.). Le format *RSS* est utilisé partout : sur des blogs, des forums, des sites d'actualité, etc. Il était donc très important de pouvoir l'exploiter correctement dans Silverlight.

Encore une fois, ce n'est pas dans Silverlight que nous allons trouver la solution mais dans C#. En effet, le *Framework 3.0* nous offre une nouveauté des plus utiles et souvent sous-exploitée : la classe *SyndicationFeed*.

À partir de ce moment-là, il devient simple de créer une application Silverlight parvenant à lire du RSS.

Notre application est composée d'un header (haut de l'application) et de deux colonnes : l'une pour afficher les différents titres du RSS et l'autre pour afficher une vue plus détaillée du contenu du RSS (de l'élément sélectionné pour être plus exact).

Pour le header, rien de très compliqué : une textbox et un bouton, à peu de choses près :

```
<StackPanel Orientation="Vertical" Width="845" >
    <TextBlock Text="Enter feed address below. If this
        control and the feed are not hosted on the same
        domain, add a clientaccesspolicy.xml or
        crossdomain.xml file to domain where
        the feed is hosted to enable
        cross-domain access."
        TextWrapping="Wrap" FontSize="14" />
    <StackPanel Orientation="Horizontal" Margin="0, 10, 0, 10">
        <TextBox Width="400" x:Name="feedAddress" FontSize="14"/>
        <Button Content="Fetch feed" Margin="20, 0, 0, 0"
            FontSize="14" Click="Button_Click" />
    </StackPanel>
</StackPanel>
```

Ensuite, pour les deux colonnes du bas qui affichent le contenu, nous avons une *ListBox* et un *StackPanel* avec un *ScrollViewer* pour afficher le texte :

```
<StackPanel Orientation="Horizontal"
    Margin="0, 10, 0, 10">
    <TextBox Width="400" x:Name="feedAddress"
        FontSize="14"/>
    <Button Content="Fetch feed" Margin="20, 0, 0, 0"
        FontSize="14" Click="Button_Click" />
</StackPanel>
<StackPanel Orientation="Horizontal">
    <ListBox x:Name="itemsList"
        ItemsSource="{Binding}" Width="325"
        Height="500"
        SelectionChanged="itemsList_SelectionChanged"
        Visibility="Collapsed">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <TextBlock
                    Text="{Binding Title.Text,
                        Converter={StaticResource htmlSanitizer}}"
                    TextWrapping="Wrap" Width="300"
                    FontSize="14"/>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
    <StackPanel x:Name="itemContent" Orientation="Vertical"
        Width="500" Height="500" Margin="20, 0, 0, 0"
        Visibility="Collapsed">
        <HyperlinkButton Content="{Binding Title.Text,
            Converter={StaticResource htmlSanitizer}}"
```

```

        NavigateUri="{Binding Links,
        Converter={StaticResource linkFormatter}}"
        FontSize="16" TargetName="_blank" />
<TextBlock Text="{Binding PublishDate}"
        TextWrapping="Wrap" FontSize="14" />
<ScrollView Height="445" Margin="0, 10, 0, 0">
    <TextBlock Text="{Binding Summary.Text,
        Converter={StaticResource htmlSanitizer}}"
        TextWrapping="Wrap" FontSize="14"/>
</ScrollView>
</StackPanel>
</StackPanel>

```

Sans oublier que nous avons besoin de deux ressources stockées au tout début de notre code dans les ressources de l'*usercontrol* :

```

<UserControl.Resources>
    <local:HtmlSanitizer x:Key="htmlSanitizer"/>
    <local:LinkFormatter x:Key="linkFormatter"/>
</UserControl.Resources>

```

Au niveau du code, rien d'extraordinaire non plus. Dans le constructeur, nous mettrons une valeur par défaut dans la *textbox* :

```

public Page()
{
    InitializeComponent();

    // Start with a default feed
    feedAddress.Text =
        "http://feeds.wired.com/wired/index?format=xml";
}

```

Sur le clic du bouton, nous allons instancier un objet *WebClient*. Cet objet ira télécharger le contenu. Nous pourrons ensuite traiter le contenu avec notre objet *SyndicationFeed*, à l'aide d'un événement.

Le code asynchrone

On voit souvent le mot *async* utilisé dans le code. En fait, il s'agit de traiter plusieurs choses en parallèle pour augmenter la vitesse d'exécution. Ce que nous faisons ici, c'est dire : *"Va rechercher le contenu et notifie-nous d'un événement quand tu as fini"*. C'est pour cela que l'événement s'appelle *client_OpenReadCompleted*, *completed* signifiant complet.


```
private void Button_Click(object sender, RoutedEventArgs e)
{
    itemsList.Visibility = Visibility.Collapsed;
    itemContent.Visibility = Visibility.Collapsed;

    // Make HTTP request to get feed
    WebClient client = new WebClient();
    client.OpenReadCompleted += new
    ➔ OpenReadCompletedEventHandler(client_OpenReadCompleted);
    client.OpenReadAsync(new Uri(feedAddress.Text));
}
```

Le résultat sera transmis en paramètre de l'événement. Nous pouvons donc le récupérer dans la fonction `client_OpenReadCompleted` qui sera appelée lorsque le téléchargement du contenu sera fini :

```
void client_OpenReadCompleted(object sender,
    OpenReadCompletedEventArgs e)
{
    if (e.Error == null)
    {
        // Load feed into SyndicationFeed
        XmlReader reader = XmlReader.Create(e.Result);
        SyndicationFeed feed = SyndicationFeed.Load(reader);

        // Set up databinding
        itemsList.DataContext = (feed as SyndicationFeed).Items;

        itemsList.Visibility = Visibility.Visible;
    }
}
```

Nous avons dans les ressources deux objets qui sont des *converters*. On utilise ces objets pour convertir une valeur. Par exemple, nous allons recevoir un texte mais il faut qu'il soit bien formaté. Pour cette raison, nous avons une classe `HtmlSanitizer` :

```
public class HtmlSanitizer : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter,
        System.Globalization.CultureInfo culture)
    {
        // Remove HTML tags and empty newlines and spaces
        string returnString =
            Regex.Replace(value as string, "<.*?>", "");
    }
}
```

```

        returnString =
            Regex.Replace(returnString, @"\n+\s+", "\n\n");

        // Decode HTML entities
        returnString =
            HttpUtility.HtmlDecode(returnString);

        return returnString;
    }

    public object ConvertBack(object value,
        Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

L'objet implémente une interface `IValueConverter`. C'est de cette manière que le programme sait que c'est un *converter*. Nous avons la même chose pour `LinkFormatter` :

```

public class LinkFormatter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        // Get the first link - that's the link to the post
        return
            ➤ ((Collection<SyndicationLink>)value).FirstOrDefault().Uri;
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

Au niveau de notre page de test *HTML*, nous avons la même chose que d'habitude :

```

<div id="Div1">
    <object data="data:application/x-silverlight-2,"
        type="application/x-silverlight-2"
        width="100%" height="100%">
        <param name="source"
            value="ClientBin/SyndicationFeedReader.xap"/>
    </object>
</div>

```

```

<param name="onerror"
      value="onSilverlightError" />
<param name="background"
      value="white" />
<param name="minRuntimeVersion"
      value="2.0.30923.0" />
<param name="autoUpgrade" value="true" />
<a href="http://go.microsoft.com/fwlink/?LinkId=124807"
  style="text-decoration: none;">
  
</a>
</object>
<iframe
  visibility="hidden" height="0" width="0" border="0px"></iframe>
</div>

```

Cet exemple court mais utile est maintenant terminé.



► Fig. 3.3 : Notre application terminée

Nous allons étudier dans le prochain chapitre les concepts avancés de Silverlight. Vous verrez aussi par la suite comment améliorer l'insertion de Silverlight lorsque vous utilisez *ASP.NET* comme technologie serveur.

3.9 Check-list

Dans ce chapitre nous avons étudié :

- ✓ l'exploitation des sources de données SQL ;
- ✓ l'utilisation des Web services ;
- ✓ LINQ ;
- ✓ la manipulation XML au travers d'un exemple ;
- ✓ la manipulation de données RSS au travers d'un exemple.

4



4.1 Introduction à ASP.NET	182
4.2 Les contrôles ASP.NET	191
4.3 Les contrôles ASP.NET pour Silverlight	198
4.4 Interaction de Silverlight avec la page	210
4.5 Check-list	211

Silverlight et ASP.NET

ASP.NET est une technologie basée sur la plateforme .NET de Microsoft qui vous permet de créer des sites web simplement. Plongez-vous dans ce chapitre si vous souhaitez connaître tout ce qu'il faut savoir sur la conception et le développement d'applications web avec ASP.NET et Silverlight.

4.1 Introduction à ASP.NET

ASP.NET a accès à toutes les classes du Framework .NET qui est un ensemble de composants pouvant être utilisés dans différents langages de programmation (grâce à la *CLR*, *Common Language Runtime*) ; les plus connus sont :

- *VB.NET* successeur de *VB* ;
- *C#*, nouveau langage spécialement conçu pour la plateforme .NET ;
- *J#* qui a été introduit spécialement pour permettre la réutilisation des blocs écrits en *J++* et attirer les développeurs Java, etc.

Il en existe bien d'autres (*Cobol*, *PHP*, ...). On notera l'arrivée de deux nouveaux, récemment :

- *IronRuby* qui, comme son nom l'indique, est un dérivé de Ruby ;
- *IronPython* qui, de la même manière, est un dérivé de Python.

En résumé, peu importe le langage utilisé, Microsoft met à votre disposition un ensemble d'outils qui vous faciliteront l'écriture d'une application ou/et d'un site Internet. L'ensemble de ces outils constitue le Framework .NET.

Nous sommes aujourd'hui à la version 3.5 du Framework .NET. La version 3.0 a été introduite avec Vista. Vous pouvez trouver la version 3.5 sur Internet (voir *Les prérequis* dans ce chapitre).

ASP.NET

ASP.NET désigne un ensemble de technologies qui vont vous simplifier l'écriture d'applications web dynamiques. *ASP.NET* succède à *ASP* mais ne vous y trompez pas, ce n'est pas du tout la même chose. Le concept a été complètement repensé.

Dans cet ensemble de technologies on trouvera, par exemple, des outils qui nous permettront de faciliter la gestion de la sécurité. En effet, sécuriser des pages et créer des espaces membres est un jeu d'enfants en *ASP.NET*. De la même manière, la gestion des données est très simple de prime abord. Si les composants *ASP.NET* ne suffisent pas, vous avez accès à tout ce que le Framework .NET vous propose. Le but ici n'est pas de décrire l'ensemble des fonctionnalités de *ASP.NET* mais de vous donner tous les outils et connaissances de base pour créer et comprendre votre premier site *ASP.NET*.

Prérequis

Les sceptiques se disent certainement qu'avec Microsoft, il va falloir payer. Mais ce n'est pas le cas. On peut bien programmer avec *ASP.NET* gratuitement. Il existe en effet une gamme *express* des produits Microsoft qui vous permet d'obtenir des outils suffisants pour la création de vos premiers projets.

Vous pouvez dès lors employer *Visual Web Developer Express* pour développer vos sites Internet et *SQL Server 2005 Express* comme base de données.

Vous trouverez ces deux outils ici : <http://msdn.microsoft.com/fr-fr/express/aa975050.aspx>.

Les outils en anglais

Si vous maîtrisez l'anglais, prenez soin de choisir les versions anglaises qui sont plus complètes au niveau du support et de la compatibilité.

Il est beaucoup plus délicat d'installer Silverlight pour Visual Studio dans sa version française que dans sa version anglaise.

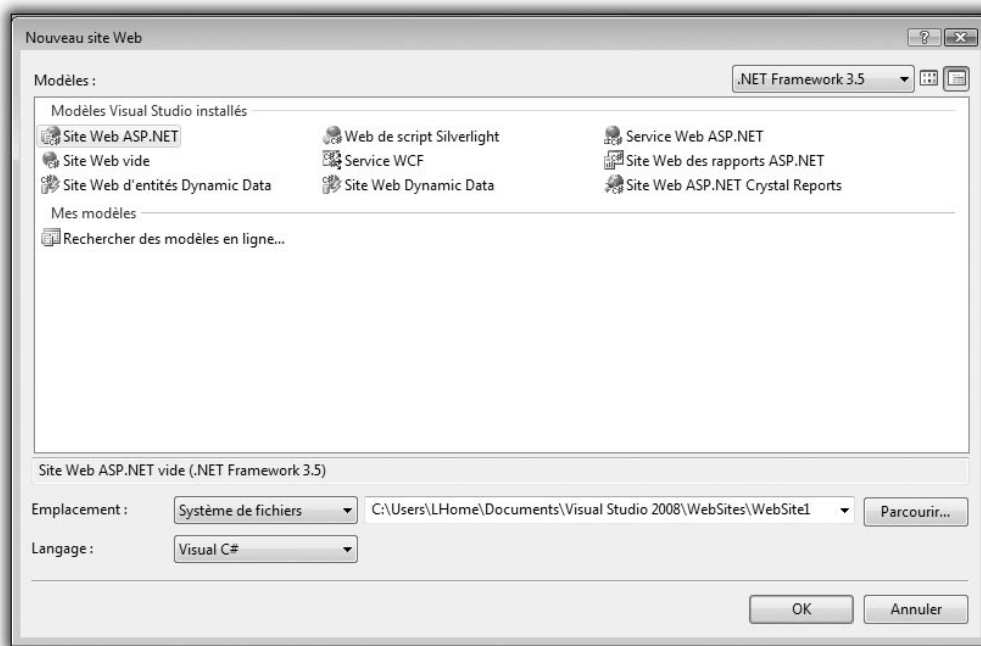
Si vous désirez directement utiliser les outils professionnels, téléchargez *Visual Studio 2008*.

Vous pouvez également télécharger le Framework 3.5 à cette adresse : <http://www.microsoft.com/downloads/details.aspx?FamilyId=333325FD-AE52-4E35-B531-508D977D32A6&displaylang=en>.

Cependant, ce n'est pas obligatoire, le Framework 2.0 installé par défaut sur tout les Windows (XP, Vista, Windows Server) est largement suffisant pour créer des applications web. Le 3.5 apportent de nombreuses nouveautés qu'il n'est pas intéressant de connaître quand on débute la programmation web avec *ASP.NET*.

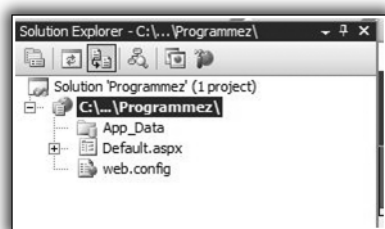
Premier exemple

Que vous possédiez *Visual Web Developer* ou *Visual Studio*, la procédure est la même. La première chose à faire est de créer un site web. Pour cela, rendez-vous dans le menu **File/New/Web Site**.



► Fig. 4.1 : Fenêtre d'application

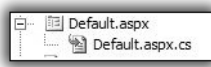
Sélectionnez **ASP.NET Web Site**, donnez-lui un nom et cliquez sur OK. Visual Studio/Web Developer crée pour vous un ensemble de fichiers et de dossiers pour vous aider à démarrer. Vous pouvez voir ces fichiers dans l'Explorateur de solution.



► Fig. 4.2 : Explorateur de solution

Default.aspx est souvent la page par défaut d'un site *ASP.NET*. Ceci est configurable dans *IIS*, le programme traitant les demandes sur des pages Internet (qui écoute sur le port 80 par exemple). C'est ce que fait Apache si vous êtes habitué à travailler avec *PHP*, *Ruby*, etc.

Default.aspx a un fichier qui lui est attaché. Il est nommé *Default.aspx.cs*.



► Fig. 4.3 :
Fichier Default.aspx

C'est dans ce fichier que se trouvera toute la logique métier de notre page. Microsoft a voulu séparer le design et le code. Vous aurez donc tout l'aspect graphique au niveau de *Default.aspx* et tout l'aspect code au niveau de *Default.aspx.cs* (on appelle le code contenu dans ce fichier le code behind). Ceci n'est pas obligatoire, vous pouvez déclarer le design et le code dans un même fichier. Le code se trouvera alors entre des balises script mais cela nuit à la lisibilité du code et à la compréhension de celui-ci.

Default.aspx ressemble à ceci lorsque vous l'ouvrez :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
➤ Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
➤ "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

            </div>
        </form>
    </body>
</html>
```

Cette page ne fait rien du tout.

Si vous voulez obtenir un aperçu dans le navigateur, cliquez du bouton droit sur le nom de la page dans l'Explorateur de solution et cliquez sur **View in browser**.

Par défaut, votre navigateur se lance et affiche une page blanche. En fait, le contenu de votre page doit être situé entre les balises form. Nous reviendrons plus tard sur ces balises.

La première ligne de notre page indique un certain nombre d'informations :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
➤ Inherits="_Default" %>
```

Cette ligne est appelée une *directive*. Toutes les directives d'une page commencent par `<%@` et finissent par `%>`. Ces directives permettent de déclarer des informations nécessaires au compilateur pour compiler la page. Par exemple, nous avons besoin de savoir quel langage est utilisé (`Language="C#"`) pour connaître le compilateur qui traitera la page ou le fichier contenant le code (`CodeFile="Default.aspx.cs"`).

Le reste de la page devrait vous être familier si vous avez l'habitude de programmer des sites Internet.

On retrouve une ligne pour le *dtd* :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Visual Studio et les DTD

Visual Studio gère bien ce *dtd* et indique une erreur lorsque vous ne le respectez pas dans votre HTML.

Le reste de la page ne devrait pas vous poser de gros problème de compréhension, mis à part le form.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

    </div>
  </form>
</body>
</html>
```

Au niveau du code attaché à notre page, nous obtenons ceci :

```
using System;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
```

```
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

Code attaché

Nous avons à chaque fois ce principe de deux fichiers. Un pour le code, l'autre pour la représentation. En effet, en Silverlight on trouve un fichier XAML et un fichier de code. En ASP.NET, nous avons un fichier HTML et un autre fichier pour le code. Ce principe est appliqué partout dans les technologies Microsoft.

On remarque directement des usings qui permettent de faciliter l'utilisation des composants du Framework .NET.

Par exemple, sans `using System.IO`, vous devriez écrire dans votre code :

```
System.IO.File.Exists("file.txt");
```

Avec le `using`, vous simplifiez votre code :

```
File.Exists("file.txt");
```

Les usings qui sont par défaut affichés dépendent du Framework que vous utilisez. Ici, nous avons employé la dernière version de celui-ci (3.5). Ne vous inquiétez pas si vous avez des usings différents des miens.

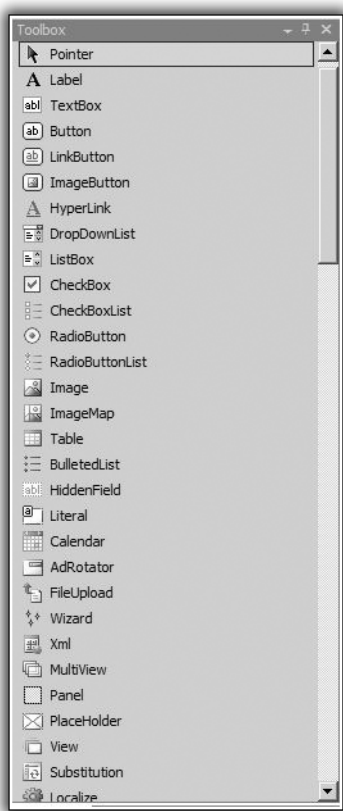
Ce qui suit est plus intéressant. On voit la déclaration d'une classe qui hérite de `Page` et déclare une méthode `Page_Load`. Cette méthode est un exemple d'*event handler*. Celle-ci déclenche l'événement `Load` de notre page ; elle sera exécutée à chaque fois que nous rechargerons la page, *Load* en anglais signifiant *Chargement*.

Il existe de nombreux événements déclenchés au chargement de la page que nous verrons un peu plus loin.

Outre toutes les balises HTML que vous connaissez, ASP.NET propose de nombreux contrôles qui vous simplifieront l'écriture de vos applications web. Ces derniers commencent tous par <asp :. On trouvera, parmi les plus utilisés :

- le *Label* ;
- le *GridView* ;
- le *Repeater* ;
- le *Literal*, etc.

Vous trouvez la liste entière de ces contrôles dans la *toolbox* de votre *Visual Studio/Web Developer*.



► Fig. 4.4 :
Barre d'outils

Pour insérer l'un de ces contrôles dans votre page, sélectionnez-le dans votre *toolbox* et déplacez-le sur votre page :

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
```

```
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" Text=" Mon premier
            label"></asp:Label>
        </div>
    </form>
</body>
</html>
```

Nous avons par exemple ajouté un label à notre page. Remarquez la propriété `runat="server"` qui permet d'indiquer que ce contrôle devra être évalué au niveau du serveur.

Si vous lancez la page de votre Explorateur, vous verrez que notre `asp:label` a été transformé en une balise `span` :

```
<span id="Label1">Mon premier label</span>
```

Que s'est-il passé ? Le compilateur a vu un contrôle qui devait être interprété par le serveur, celui-ci l'a traité pour le remplacer par du HTML, compréhensible par le navigateur.

Dans ce cas, l'utilisation d'un contrôle ne s'avère pas utile. L'utilité vient lorsque vous traitez ce contrôle dans le *code behind*. Vous avez en effet la possibilité d'éditer les propriétés du label en *code behind* (`Text`, etc.) :

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = "Ce label a été traité dans le code behind";
}
```

Cela donnera le résultat suivant :

```
<span id="Label1">Ce label a été traité dans le code behind</span>
```

i Code attaché

Ce que vous écrivez dans le code behind pour la propriété **Text** prend le dessus sur ce que vous avez écrit dans le design de la page car l'événement **Load** de la page passe après la traitement du *render* de la page. La propriété de votre label est dans un premier temps égale à **Mon premier label** mais elle est ensuite réécrite au moment du *Load*.

Le Web.config

Le *Web.config* est un fichier que vous retrouvez à la racine de votre application et qui contient toutes les informations sur la configuration de celle-ci. Si vous ouvrez le *Web.config* de votre application web, vous trouverez une série d'informations. Parmi les informations importantes :

La déclaration des éléments qui se trouveront dans le Web.config

```
<configSections>
  <sectionGroup name="system.web.extensions"
    type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
    System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=31BF3856AD364E35">
    <sectionGroup name="scripting"
      type="System.Web.Configuration.ScriptingSectionGroup,
      System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
      PublicKeyToken=31BF3856AD364E35">
      <section name="scriptResourceHandler"
        type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
        System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
        PublicKeyToken=31BF3856AD364E35" requirePermission="false"
        allowDefinition="MachineToApplication"/>
      ...
    </sectionGroup>
  </sectionGroup>
</configSections>
```

Les différentes libraires dont à besoin notre application

```
<assemblies>
  <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=B77A5C561934E089"/>
  <add assembly="System.Web.Extensions, Version=3.5.0.0,
    Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
    Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
</assemblies>
```

La déclaration des contrôles ASP.NET

```
<pages>
  <controls>
    <add tagPrefix="asp" namespace="System.Web.UI"
      assembly="System.Web.Extensions, Version=3.5.0.0,
      Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
      assembly="System.Web.Extensions, Version=3.5.0.0,
      Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  </controls>
</pages>
```

La déclaration de nos chaînes de connexion vers une base de données

```
<connectionStrings>
  <add name="LimbourgBDConnectionString" connectionString="Data
  Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\ProgrammezBD.mdf;
  Integrated Security=True;User Instance=True"
  providerName="System.Data.SqlClient"/>
</connectionStrings>
```

4.2 Les contrôles ASP.NET

Microsoft vous fournit des contrôles que vous pouvez utiliser pour enrichir vos applications web. On peut répertorier ces contrôles en plusieurs catégories.

Les contrôles standard

Ces contrôles vous permettent d'afficher des éléments standard : boutons, labels, champs de texte, etc. On pourra par exemple facilement créer un formulaire simple à l'aide de ces contrôles :

```
<asp:Label ID="lbNom" runat="server" Text="Nom"></asp:Label> :
  <asp:TextBox ID="tbNom" runat="server"></asp:TextBox> <br />
  <asp:Label ID="lbPrenom" runat="server"
  Text="Prenom"></asp:Label> :
  <asp:TextBox ID="tbPrenom" runat="server"></asp:TextBox> <br />
  <asp:Label ID="lbEmail" runat="server" Text="Email"></asp:Label> :
  <asp:TextBox ID="tbEmail" runat="server"></asp:TextBox> <br />
  <asp:Label ID="lb" runat="server" Text="J'ai plus de 18 ans">
  </asp:Label> :
  <asp:CheckBox ID="cbPlusDe18ans" Text="" runat="server" /><br />
  <asp:Button ID="btGo" runat="server" Text="Envoyer" />
```

Ceci sera affiché sur votre navigateur Internet :

The image shows a web form with the following elements:

- Label: Nom, followed by a text input field.
- Label: Prenom, followed by a text input field.
- Label: Email, followed by a text input field.
- Label: J'ai plus de 18 ans, followed by a checkbox.
- Button: Envoyer.

► Fig. 4.5 :
Contrôles de base d'ASP.NET

On peut regarder ce que cela génère au niveau du code source :

```
<span id="lbNom">Nom</span> :
<input name="tbNom" type="text" id="tbNom" /> <br />
<span id="lbPrenom">Prenom</span> :
<input name="tbPrenom" type="text" id="tbPrenom" /> <br />
<span id="lbEmail">Email</span> :
<input name="tbEmail" type="text" id="tbEmail" /> <br />
<span id="lb">J'ai plus de 18 ans</span> :
<input id="cbPlusDe18ans" type="checkbox" name="cbPlusDe18ans" /><br />
<input type="submit" name="btGo" value="Envoyer" id="btGo" />
```

Les contrôles de validation

ASP.NET vous fournit un ensemble de contrôles qui vous permettront d'effectuer des vérifications sur les données entrées par les utilisateurs. On peut donc améliorer notre formulaire en ajoutant un contrôle pour vérifier si l'email n'est pas nul :

```
<asp:Label ID="lbNom" runat="server" Text="Nom"></asp:Label> :
<asp:TextBox ID="tbNom" runat="server"></asp:TextBox> <br />
<asp:Label ID="lbPrenom" runat="server" Text="Prenom"></asp:Label> :
<asp:TextBox ID="tbPrenom" runat="server"></asp:TextBox> <br />
<asp:Label ID="lbEmail" runat="server" Text="Email"></asp:Label> :
<asp:TextBox ID="tbEmail" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator
    ID="ValEmail" ControlToValidate="tbEmail" runat="server"
    ErrorMessage="Email Requis"></asp:RequiredFieldValidator> <br />
<asp:Label ID="lb" runat="server" Text="J'ai plus de 18 ans">
</asp:Label> :
<asp:CheckBox ID="cbPlusDe18ans" Text="" runat="server" /><br />
<asp:Button ID="btGo" runat="server" Text="Envoyer" />
```

Il existe plusieurs contrôles de ce type :

- *RequiredFieldValidator* ;
- *RangeValidator* ;

- *RegularExpressionValidator* ;
- *CompareValidator* ;
- *CustomValidator*.

Vous pouvez également créer vos propres contrôles de validation.

Les contrôles riches

ASP.NET vous propose quelques contrôles avec une réelle plus-value pour vous. Par exemple, créer un calendrier dynamique en *ASP.NET* est vraiment très simple. Il existe en fait un contrôle asp :calendar qui vous permet d'afficher un calendrier dynamique. Dans ce type de contrôles, on retrouve aussi la possibilité de créer des *wizards* facilement (exemple : création de formulaires en plusieurs étapes).

Voici un exemple de calendrier en *ASP.NET* :

```
<asp:Calendar id="Calendar1" runat="server" BorderStyle="Double"
  BorderWidth="3px" DayNameFormat="FirstTwoLetters"
  FirstDayOfWeek="Monday" ShowGridLines="True" CellPadding="0"
  SelectedDate="2004-05-03">
  <OtherMonthDayStyle ForeColor="LightGray">
  </OtherMonthDayStyle>
</asp:Calendar>
```

Les contrôles de données

Les contrôles de données permettent d'aller rechercher facilement des données dans une base ou un fichier XML.

Vous utiliserez ces contrôles pour ce qui concerne la visualisation, l'ajout, la modification ou la suppression des données.

Vous pourrez par exemple aisément afficher les données dans un tableau à l'aide d'un *GridView* et un *DataSource* (*ObjectDataSource*, *SQLDataSource* ou *XMLDataSource* en fonction de la source des données).

Framework 3.5 et ASP.NET 3.5

Le Framework 3.5 a apporté son lot de nouveautés à ce niveau avec l'ajout du *ListView*, du *DataPager* et du *LINQDataSource*.

Ci-après, un exemple d'utilisation du contrôle *repeater* et *SQLDataSource* :

```
<asp:Repeater ID="Repeater3" runat="server" DataSourceID="DSCC">
  <ItemTemplate>
    <h1><%# Eval("Prenom") %> <%# Eval("Nom") %></h1>
    <p><%# Eval("Titre") %> - <%# Eval("Biographie") %></p>
  </ItemTemplate>
</asp:Repeater>
<asp:SqlDataSource ID="DSCC" runat="server"
  ConnectionString="<%= $ ConnectionStrings.LimbourgBDConnectionString %>"
  SelectCommand="SELECT TOP 3 [PersonneId], [Nom], [Prenom],
    [Biographie], [Titre] FROM [Personne] WHERE ([PersonneId] =
    @PersonneId)">
  <SelectParameters>
    <asp:QueryStringParameter DefaultValue="1" Name="PersonneId"
      QueryStringField="id" Type="Int32" />
  </SelectParameters>
</asp:SqlDataSource>
```

Ces quelques lignes vont permettre d'afficher une liste de personnes. Eval permet de mapper un élément avec un champ de la base de données. Le *repeater* est comme une boucle qui parcourt tous les éléments de la source de données passée comme valeur de la propriété DataSourceID.

Ici la source de données contient les résultats de la requête SQL passée comme valeur à la propriété SelectCommand :

```
SELECT TOP 3 [PersonneId], [Nom], [Prenom], [Biographie], [Titre] FROM
[Personne] WHERE ([PersonneId] = @PersonneId)
```

@PersonneId désigne un paramètre que nous définissons juste après :

```
<SelectParameters>
  <asp:QueryStringParameter DefaultValue="1" Name="PersonneId"
    QueryStringField="id" Type="Int32" />
</SelectParameters>
```

On indique que le paramètre doit être trouvé dans l'URL de la requête. On va chercher le paramètre id de l'URL. S'il n'existe pas, on prend par défaut la valeur 1.

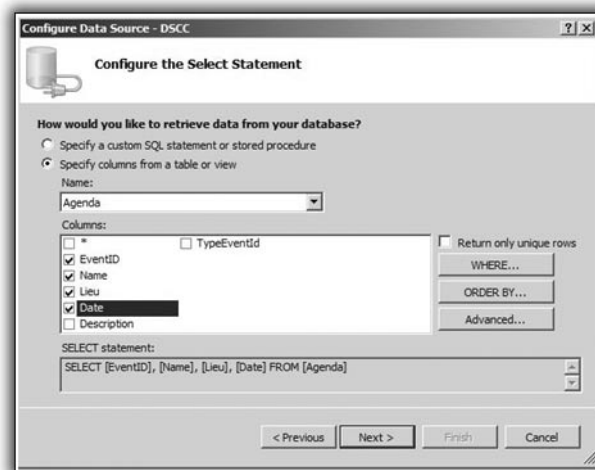
Pour simplifier l'écriture de ce code, Visual Studio vous fournit un *wizard* de paramétrage d'un *DataSource*. Pour utiliser ce dernier, rendez-vous en mode Design et cliquez sur la petite icône en haut à droite de votre *SQLDataSource*.



► Fig. 4.6 :
Datasource

Cliquez sur **Configure DataSource**. Une nouvelle fenêtre apparaît ; vous avez la possibilité de définir votre chaîne de connexion à votre base de données. Cette chaîne peut être définie en cliquant sur **New Connection** et en remplissant les différents champs de connexion (nom du serveur, login, mot de passe, base de données).

En cliquant sur **Suivant**, un utilitaire vous permet de créer votre requête.



► Fig. 4.7 :
Éditeur de requêtes

Cliquez sur **Suivant** puis sur **Terminer**. Votre *DataSource* est configuré.

Pour en savoir plus sur *SQLDataSource*, consultez le lien : <http://msdn.microsoft.com/fr-fr/library/system.web.ui.webcontrols.sqldatasource.aspx>.

Les contrôles de navigation

Ces contrôles vous permettent d'afficher des éléments de navigation standard : menus, arbres d'éléments. En effet, on peut facilement afficher un arbre d'éléments avec le contrôle *asp:TreeView* :

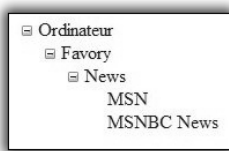
```
<asp:TreeView ID="TreeView1" runat="server">
  <Nodes>
    <asp:TreeNode Text="Ordinateur">
      <asp:TreeNode Text="Favory">
        <asp:TreeNode Text="News">
```

```

        <asp:TreeNode Text="MSN" NavigateUrl="http://www.msn.com"/>
        <asp:TreeNode Text="MSNBC News"
            NavigateUrl="http://www.msnbc.msn.com"/>
        </asp:TreeNode>
    </asp:TreeNode>
</Nodes>
</asp:TreeView>

```

On obtient ceci :



► Fig. 4.8 :
Vue d'un arbre de données

Vous pouvez également mapper le contenu de votre *TreeView* à un fichier XML, par exemple. Pour plus d'information sur les contrôles de navigation : <http://quickstarts.asp.net/QuickStartv20/aspnet/doc/ctrlref/navigation/default.aspx>.

Les contrôles de login

Ces contrôles sont très souvent utilisés pour tout ce qui est *gestion utilisateur*. Vous pouvez facilement gérer des comptes utilisateur avec *ASP.NET*. Celui-ci vous fournit les contrôles pour les actions suivantes :

- login d'un utilisateur ;
- affichage du nom de l'utilisateur actuellement connecté ;
- enregistrement d'un utilisateur ;
- perte de mots de passe ;
- changement de mot de passe ;
- affichage d'éléments en fonction du statut de l'utilisateur (connecté, non connecté, dans un groupe, ou pas) ;
- gestion de rôles d'utilisateur (administrateur, membre, etc.).

Pour permettre l'utilisation de ces composants, vous devez régler ces derniers à l'aide de l'administration centrale de votre site *ASP.NET* :



► Fig. 4.9 :
Centrale d'administration
d'ASP.NET

Dans l'onglet **Sécurité**, vous pouvez choisir d'utiliser *ASP.NET Membership Provider* en cliquant sur le lien *Sélectionnez le type d'authentification*. Sélectionnez *Par internet*.

Pour plus d'informations sur les composants *login*, consultez le lien suivant : <http://quickstarts.asp.net/QuickStartv20/aspnet/doc/ctrlref/login/default.aspx>.

Les contrôles HTML

Vous pouvez utiliser les contrôles HTML que vous avez l'habitude d'employer et interagir avec eux dans le *code behind* en leur ajoutant la propriété `runat=server` :

```
Entrez votre nom : <input id="Name" type="text" size=40 runat=server>
```

Vous pouvez alors modifier les propriétés de ces contrôles dans le code behind :

```
Name.Value = "ici";
```

Lorsque *ASP.NET* voit un tag HTML avec la propriété `runat`, il reconnaît un contrôle *ASP.NET*. Par exemple, ici, il sait que l'`input` est un `HtmlInputText`.

Plus d'informations ici : <http://quickstarts.asp.net/QuickStartv20/aspnet/doc/ctrlref/html/default.aspx>.

Postback et ViewState

Voici une petite introduction à ce qui caractérise le modèle de programmation *ASP.NET*. Revenons sur notre exemple de formulaire. Lorsque nous voulons envoyer les informations de notre formulaire, nous devons gérer un événement sur le clic du bouton. Pour cela, nous lui déclarons un handler :

```
<asp:Button ID="btGo" runat="server" Text="Envoyer" onclick="btGo_Click" />
```

Lorsque vous cliquez sur le bouton, il y a un *Postback* ou une requête *POST* sur la page en cours. Vous devez gérer cet événement dans le *code behind* sur la méthode `btGo_Click` :

```
protected void btGo_Click(object sender, EventArgs e)
{
    // Traitement ici
}
```

En HTML, l'élément `form` permet d'envoyer une requête *POST* (ou *GET*). Si on regarde le code source de notre page, on voit que notre `form runat=server` s'est transformé en formulaire :

```
<form name="form1" method="post" action="Default.aspx"
  onsubmit="javascript:return WebForm_OnSubmit();" id="form1">
  ...
</form>
```

L'action du formulaire est bien la page actuelle.

Le JavaScript est là car il y a des choses à vérifier après l'envoi du formulaire (à cause de notre *validator*).

On remarque également un champ de type *input* appelé *ViewState* :

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
  value="/wEPDwUKLTcwNjkwMjI1M2QYAUeX19Db250cm9sc1JlcXVpcmVQb3N0QmFja
  0tleV9fFgEFdWNIUEExlc0RlMThhbnOWq/eV0Z6qaaIUowq3lP+t4xSruA==" />
```

Ce champ *hidden* est géré automatiquement par *ASP.NET* pour sauvegarder l'état des différents contrôles de la page. C'est grâce à cela que lorsque vous effectuez un *PostBack*, vos informations sont toujours visibles dans le formulaire. Vous n'avez donc pas besoin de sauver vous-même l'état des différents éléments de votre page en *ASP.NET*.

4.3 Les contrôles ASP.NET pour Silverlight

Avec l'arrivée et l'utilisation de plus en plus massive de *Silverlight*, Microsoft a dû réfléchir pour faciliter l'utilisation de *Silverlight* dans *ASP.NET*. En effet, insérer *Silverlight* comme nous l'avons vu auparavant dans le livre n'est pas une bonne idée.

Voici la structure HTML d'une page avec *Silverlight* à l'intérieur :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<!-- saved from url=(0014)about:internet -->
<head>
  <title>Mastermind</title>

  <style type="text/css">
html, body {
  height: 100%;
  overflow: auto;
}
body {
  padding: 0;
  margin: 0;
}
```

```

#SilverlightControlHost {
    height: 100%;
}
</style>

<script type="text/javascript">
    function onSilverlightError(sender, args) {

        var appSource = "";
        if (sender != null && sender != 0) {
            appSource = sender.getHost().Source;
        }
        var errorType = args.ErrorType;
        var iErrorCode = args.ErrorCode;

        var errMsg = "Unhandled Error in Silverlight 2 Application
➡ " + appSource + "\n" ;

        errMsg += "Code: " + iErrorCode + "      \n";
        errMsg += "Category: " + errorType + "      \n";
        errMsg += "Message: " + args.ErrorMessage + "      \n";

        if (errorType == "ParserError")
        {
            errMsg += "File: " + args.xamlFile + "      \n";
            errMsg += "Line: " + args.lineNumber + "      \n";
            errMsg += "Position: " + args.charPosition + "      \n";
        }
        else if (errorType == "RuntimeError")
        {
            if (args.lineNumber != 0)
            {
                errMsg += "Line: " + args.lineNumber + "      \n";
                errMsg += "Position: " + args.charPosition + "      \n";
            }
            errMsg += "MethodName: " + args.methodName + "      \n";
        }

        throw new Error(errMsg);
    }
</script>
</head>

<body>
    <!-- Les erreurs d'exécution Silverlight s'afficheront ici.
    Il s'agit d'informations de débogage qui doivent être supprimées ou
    masquées, une fois le débogage terminé -->
    <div id='errorLocation' style="font-size: small;color: Gray;"></div>

```

```

<div id="SilverlightControlHost">
  <object data="data:application/x-Silverlight,"
    type="application/x-Silverlight-2" width="100%" height="100%">
    <param name="source" value="Mastermind.xap"/>
    <param name="onerror" value="onSilverlightError" />
    <param name="background" value="white" />
    <param name="minRuntimeVersion" value="2.0.31005.0" />
    <param name="autoUpgrade" value="true" />
    <a href="http://go.microsoft.com/fwlink/?LinkID=124807"
      style="text-decoration: none;">
      
    </a>
  </object>
  <iframe
    ➤ style='visibility:hidden;height:0;width:0;border:0px'></iframe>
</div>
</body>
</html>

```

On remarque différents éléments :

- Ce n'est pas dans la logique de *ASP.NET* d'utiliser des contrôles HTML s'initialisant avec JavaScript.
- Cette page ne ressemble en rien à ce que nous avons l'habitude de voir en *ASP.NET*.
- La page a perdu beaucoup de lisibilité par rapport à *ASP.NET*.

D'autre part, nous avons vu dans un autre chapitre ce que générerait *Expression Encoder* :

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">
<!-- saved from url=(0014)about:internet -->
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script type='text/javascript' src="MicrosoftAjax.js"></script>
<script type='text/javascript' src="Silverlight.js"></script>
<script type='text/javascript' src="SilverlightControl.js"></script>
<script type='text/javascript' src="SilverlightMedia.js"></script>
<script type='text/javascript' src="ExpressionPlayer.js"></script>
<script type='text/javascript' src="PlayerStrings.js"></script>
<script type='text/javascript' src="player.js"></script>
<script type='text/javascript' src="StartPlayer.js"></script>
<title></title>
<style type="text/css">
  html, body { margin: 0; padding: 0; height:100% }

```



```
#divPlayer_0 { min-height: 100%; height:100%; }
</style>
</head>

<body style="background-color:black;margin:0,0,0,0;overflow:auto;">
  <div id="divPlayer_0">
    <script type='text/javascript'>
      var player = new StartPlayer_0();
    </script>
  </div>
</body>
</html>
```

Même chose ici, il est très rare de voir l'insertion de fichier JavaScript comme cela dans *ASP.NET*.

Pour remédier aux deux problèmes, Microsoft a introduit deux nouveaux contrôles dans *ASP.NET* :

- *Silverlight* ;
- *MediaPlayer*.

Le contrôle MediaPlayer

Le contrôle serveur *ASP.NET MediaPlayer* vous permet d'intégrer des sources de média telles que des éléments audio (WMA) et vidéo (WMV) dans une application web, sans qu'il soit nécessaire de disposer de connaissances en XAML ou en JavaScript. Le contrôle *MediaPlayer* peut utiliser des apparences pré générées, ou vous pouvez créer des apparences personnalisées. Par exemple, vous pouvez référencer un document XAML personnalisé généré via Microsoft Expression Encoder et prenant en charge des légendes, des chapitres et des marqueurs dans la source du média. Lorsque vous configurez le contrôle *MediaPlayer* pour référencer une apparence pré générée, le document XAML associé est copié dans le projet. La propriété *MediaSkinSource* du contrôle *MediaPlayer* est également configurée pour pointer sur l'apparence référencée.

L'interaction entre le contrôle *MediaPlayer* et la page peut se faire à l'aide de *JavaScript*.

Ce lecteur *ASP.NET* s'utilise très facilement. Vous devez référencer un fichier à jouer (par exemple, un fichier *.wmv*, *.wma* ou *.mp3*), puis vous sélectionnez une apparence intégrée. Le contrôle serveur *MediaPlayer* reconnaît les formats de média pris en charge par le *plug-in Silverlight* :

```
<asp:MediaPlayer
  ID="MediaPlayer1"
```

```

    runat="server"
    MediaSource="~/Media/video.wmv"
    MediaSkinSource="~/MediaSkins/Professional.xaml"
    ScaleMode="Stretch"
    AutoLoad="true"
    AutoPlay="false"
    PluginBackground="Black"
    Height="240px"
    Width="320px">
</asp:MediaPlayer>

```

Pour fonctionner, le *MediaPlayer* a besoin d'un contrôle *ASP.NET* *ScriptManager*. Ce contrôle permet la gestion des éléments *JavaScript*.

Dans une page *ASP.NET* qui utilise le JavaScript, le *ScriptManager* charge les scripts JavaScript nécessaires au bon fonctionnement de la page :

```

<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="WebApplication1._Default" %>

<%@ Register Assembly="System.Web.Silverlight"
    Namespace="System.Web.UI.SilverlightControls"
    TagPrefix="asp" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:MediaPlayer ID="MediaPlayer1" runat="server" Height="240px"
                Width="320px"
                MediaSource="~/Lake.wmv">
            </asp:MediaPlayer>
            <br />
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
        </div>
    </form>
</body>
</html>

```

Si vous exécutez ce code et que vous regardiez le code source, vous trouverez énormément de JavaScript, notamment pour la création du lecteur :

```
<span id="MediaPlayer1_parent"></span>
<script type="text/javascript">
//
Sys.UI.Silverlight.Control.createObject('MediaPlayer1_parent',
'\u003cobject type="application/x-Silverlight" id="MediaPlayer1"
style="height:240px;width:320px;"&gt;\r\n\t\u003c
href="http://go2.microsoft.com/fwlink/?LinkID=114576&amp;v=1.0"&gt;\u003cimg
src="http://go2.microsoft.com/fwlink/?LinkID=108181"
alt="Téléchargez Microsoft Silverlight" style="border-width:0;" /&gt;
\u003c/a&gt;\r\n\u003c/object&gt;');
//]]&gt;
&lt;/script&gt;</pre>
</div>
<div data-bbox="194 404 460 577" data-label="Image">
<img alt="A screenshot of a Silverlight media player control. It displays a video of a large rock in the middle of a body of water with mountains in the background. The player has a standard interface with play, stop, and volume buttons at the bottom."/>
</div>
<div data-bbox="474 400 620 435" data-label="Caption">
<p>► Fig. 4.10 :<br/>Lecteur Silverlight</p>
</div>
<div data-bbox="152 596 902 656" data-label="Text">
<p>Le contrôle <i>MediaPlayer</i> permet de faire exactement la même chose que ce que vous aviez fait avec <i>Expression Encoder</i>. Il en va de même pour la gestion des chapitres. Vous pouvez facilement ajouter des chapitres à votre vidéo :</p>
</div>
<div data-bbox="157 671 885 901" data-label="Text">
<pre>&lt;%@ Page Language="C#" %&gt;
&lt;%@ Register Assembly="System.Web.Silverlight"
Namespace="System.Web.UI.SilverlightControls"
TagPrefix="asp" %&gt;

&lt;!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"&gt;
&lt;html &gt;
&lt;head id="Head1" runat="server"&gt;
&lt;title&gt;ASP.NET Controls for Silverlight&lt;/title&gt;
&lt;link href="~/SilverlightStyles.css" type="text/css" rel="Stylesheet" /&gt;
&lt;/head&gt;
&lt;body&gt;
&lt;form id="form1" runat="server"&gt;</pre>
</div>
<div data-bbox="844 958 900 975" data-label="Page-Footer">203</div>
```

```

<asp:ScriptManager ID="ScriptManager1" runat="server"
  EnablePartialRendering="false" />
<div>
  <asp:MediaPlayer runat="server" ID="MediaPlayer1"
    ScaleMode="Stretch" AutoPlay="true"
    MediaSource="../media/expressionstudio.wmv" Height="480"
    Width="640"
    PluginBackground="Black"
    MediaSkinSource="../skins/Professional.xaml">
    <Chapters>
      <asp:MediaChapter
        ThumbnailSource="../media/MarkerThumb 00.00.00.jpg"
        Position="0.0" Title="Chapter 1" />
      <asp:MediaChapter
        ThumbnailSource="../media/MarkerThumb 00.00.10.jpg"
        Position="10" Title="Chapter 2" />
      <asp:MediaChapter
        ThumbnailSource="../media/MarkerThumb 00.00.24.jpg"
        Position="24" Title="Chapter 3" />
    </Chapters>
  </asp:MediaPlayer>
</div>
</form>
</body>
</html>

```

L'élément `Chapters` que vous ajoutez permet de définir des chapitres dans votre vidéo.

Information en provenance de votre base de données

Il se peut que certaines informations proviennent de votre base de données comme les chapitres et l'emplacement sur fichier de votre vidéo. Dans ce cas, vous pouvez créer un Web service qui expose sous forme XML les données de la vidéo. En effet, avec la propriété `MediaDefinition` vous pouvez donner l'emplacement d'un fichier XML qui définit votre vidéo en respectant le modèle suivant :

```

<mediaDefinition>
  <mediaItems>
    <mediaItem mediaSource="video.wmv"
      placeholderSource="image0.jpg">
      <chapters>
        <chapter position="5.00"
          thumbnailSource="image1.jpg"
          title="marker1" />
        <chapter position="10.00"
          thumbnailSource="image2.jpg"
          title="marker2" />
      </chapters>
    </mediaItem>
  </mediaItems>
</mediaDefinition>

```

```

    </chapters>
  </mediaItem>
</mediaItems>
</mediaDefinition>

```

Comme indiqué précédemment dans ce chapitre, vous pouvez interagir avec le lecteur Silverlight via JavaScript.

MediaPlayer et JavaScript

Le JavaScript permet d'interagir de manière poussée avec le lecteur *Silverlight*. Vous pouvez par exemple créer sur la page d'autres boutons qui permettent d'effectuer les actions pause ou play. Ces boutons contactent alors une fonction JavaScript connectée au lecteur.

On peut aussi mettre à jour certains éléments de la page, par exemple savoir quand un chapitre change ou quand le statut change. Vous pouvez ainsi déclarer une série d'événements dans votre lecteur :

```

<asp:MediaPlayer runat="server" ID="MediaPlayer1" Width="400"
  Height="300" ScaleMode="Stretch"
    MediaSource="../media/expressionstudio.wmv" Volume="1.0"
    PlaceholderSource="../media/placeholder.JPG"
    OnClientMediaOpened="onMediaOpened"
    OnClientChapterStarted="onChapterChanged"
    OnClientMarkerReached="onMarkerReached"
    OnClientCurrentStateChanged="onStateChanged"
    OnClientVolumeChanged="onVolumeChanged">
  <Chapters>
    <asp:MediaChapter
      ➤ ThumbnailSource="../media/ExpressionStudio_
      ➤ MarkerThumb 00.00.00.jpg" Position="0.0"
      Title="Opening credits and movie start." />
    <asp:MediaChapter
      ➤ ThumbnailSource="../media/ExpressionStudio_
      ➤ MarkerThumb 00.00.10.2390000.jpg"
      ➤ Position="10.2390000"
      Title="Designing and selecting." />
    <asp:MediaChapter
      ➤ ThumbnailSource="../media/ExpressionStudio_
      ➤ MarkerThumb 00.00.24.1360000.jpg"
      ➤ Position="24.1360000"
      Title="Producing designs." />
    <asp:MediaChapter
      ➤ ThumbnailSource="../media/ExpressionStudio_
      ➤ MarkerThumb 00.00.37.9290000.jpg"
      ➤ Position="37.9290000"

```

```

        Title="Checking inventory and orders." />
        <asp:MediaChapter
        ➤ ThumbnailSource="../media/ExpressionStudio_
        ➤ MarkerThumb 00.00.52.3490000.jpg"
        ➤ Position="52.3490000"
        Title="Purchasing." />
        <asp:MediaChapter
        ➤ ThumbnailSource="../media/ExpressionStudio MarkerThumb
        ➤ 00.00.58.6180000.jpg" Position="58.6180000"
        Title="Reviewing." />
        <asp:MediaChapter
        ➤ ThumbnailSource="../media/ExpressionStudio_
        ➤ MarkerThumb 00.01.14.0820000.jpg"
        ➤ Position="74.0820000"
        Title="End credits." />
        <asp:MediaChapter
        ➤ ThumbnailSource="../media/ExpressionStudio_
        ➤ MarkerThumb 00.01.22.9640000.jpg"
        ➤ Position="82.9640000"
        Title="Silverlight end." />
    </Chapters>
</asp:MediaPlayer>

```

On voit ici la déclaration de plusieurs événements :

- OnClientMediaOpened ;
- OnClientChapterStarted ;
- OnClientMarkerReached, etc.

À chacun de ces événements correspond une fonction JavaScript, comme ici dans le changement de chapitre qui modifie le contenu HTML d'une zone HTML :

```

function onChapterChanged(sender, args) {
    // When the chapters change, set the chapter image and text in
    ➤ markup.
    var chapter = __player.get_currentChapter();
    if (chapter === -1) {
        $get('ChapterImage').src = "../media/placeholder.jpg";
        $get('ChapterIndex').innerHTML = "(none)";
    }
    else {
        $get('ChapterImage').src =
            __player.get_currentChapter().get_thumbnailSource();
        $get('ChapterIndex').innerHTML = "#" +
        ➤ (__player.get_chapterStarted() + 1) + " - " +
        ➤ __player.get_currentChapter().get_title();
        $get('MovieTime').innerHTML = __player.get_position();
    }
}

```

```
}
```

Si on veut contrôler le lecteur (play, pause, stop), nous pouvons également le faire en JavaScript :

```
function onPlay() {
    __player.play();
    $get('pause').disabled = "";
    $get('stop').disabled = "";
    $get('play').disabled = "disabled";
}
function onPause() {
    __player.pause();
    $get('pause').disabled = "disabled";
    $get('play').disabled = "";
}
function onStop() {
    __player.stop();
    $get('pause').disabled = "disabled";
    $get('stop').disabled = "disabled";
    $get('play').disabled = "";
}
```

C'est à peu près tout ce que vous pouvez effectuer avec le contrôle *MediaPlayer*.

Le contrôle Silverlight

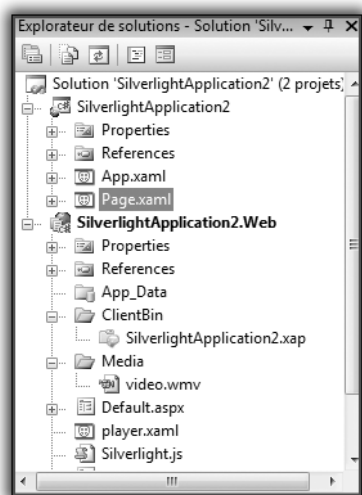
Grâce à ce contrôle, vous pouvez intégrer du code XAML et tout code de prise en charge (un assembly de code managé, un module de script de langage dynamique managé ou des bibliothèques JavaScript client, voir annexe) dans votre site web.

Code managé

Le code managé est un code compilé. À l'inverse d'un code interprété, le code compilé est organisé de manière intelligente. Plus d'information : <http://en.wikipedia.org/wiki/Compiler>.

Lorsque vous créez un projet Silverlight dans *Visual Studio*, vous avez directement la possibilité de définir un projet ASP.NET associé. Dès que cette connexion entre les deux projets est créée, vous voyez apparaître un fichier *xap* dans le dossier *ClientBin*.

Le contrôle Silverlight a besoin de ce fichier *xap*. C'est ce dernier qui va être interprété.



► Fig. 4.11 :
Explorateur de solution

Si vous créez un rectangle en *XAML*, que vous compilez votre application (**(Ctrl)+(Maj)+(B)**) et que vous utilisiez le fichier *xap* dans votre contrôle *Silverlight*, vous obtenez le résultat escompté :

```
<UserControl x:Class="SilverlightApplication2.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
    <Rectangle Height="83" Margin="68,50,137,0" VerticalAlignment="Top"
      Stroke="#FF000000">
      <Rectangle.Fill>
        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
          <GradientStop Color="#FF000000"/>
          <GradientStop Color="#FFA31212" Offset="1"/>
        </LinearGradientBrush>
      </Rectangle.Fill>
    </Rectangle>
  </Grid>
</UserControl>
```

Au niveau de la page *HTML/ASP.NET* :

```
<%@ Page Language="C#" AutoEventWireup="true" %>
```



```

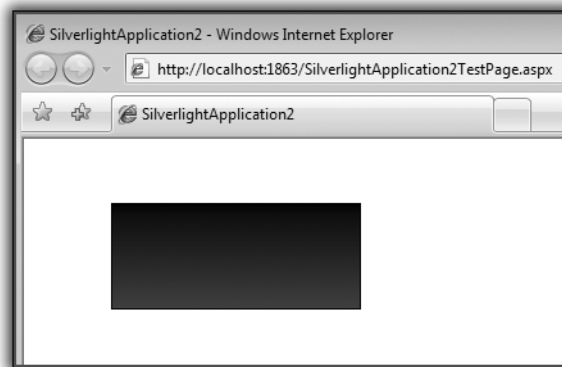
<%@ Register Assembly="System.Web.Silverlight"
    Namespace="System.Web.UI.SilverlightControls"
    TagPrefix="asp" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%;">
<head runat="server">
    <title>SilverlightApplication2</title>
</head>
<body style="height:100%;margin:0;">
    <form id="form1" runat="server" style="height:100%;">
        <asp:ScriptManager ID="ScriptManager1"
            runat="server"></asp:ScriptManager>
        <div style="height:100%;">
            <asp:Silverlight ID="Xaml1" runat="server"
                Source="~/ClientBin/SilverlightApplication2.xap"
                MinimumVersion="2.0.31005.0" Width="100%" Height="100%" />
        </div>
    </form>
</body>
</html>

```

Résultat :



► Fig. 4.12 :
Résultat de notre
application

4.4 Interaction de Silverlight avec la page

Silverlight et le code C# derrière permettent d'interagir avec les éléments de la page *HTML* (aussi appelé *DOM*).

Avant d'entamer cette partie, lisez l'annexe 2, Introduction au C#.

Dans le code attaché, vous pouvez ainsi rechercher un élément dans la page et interagir avec celui-ci ou même ajouter des événements sur ce bouton :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Browser;

namespace SilverlightApplication2
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();

            HtmlElement button =
                HtmlPage.Document.GetElementById("HTMLButtonA");
            if (button != null)
                button.AttachEvent("onclick",
                    new EventHandler<HtmlEventArgs>(
                        delegate(object o, HtmlEventArgs e) {
                            HtmlButtonClicked("A");
                        }));

            button =
                HtmlPage.Document.GetElementById("HTMLButtonB");
            if (button != null)
                button.AttachEvent("onclick",
                    new EventHandler<HtmlEventArgs>(
                        delegate(object o, HtmlEventArgs e) {
                            HtmlButtonClicked("B");
                        }));
        }
    }
}
```

```

        }));

        button =
        HtmlPage.Document.GetElementById("HTMLButtonC");
        if (button != null)
            button.AttachEvent("onclick",
                new EventHandler<HtmlEventArgs>(
                    delegate(object o, HtmlEventArgs e)
                    {
                        HtmlButtonClicked("C");
                    }
                ));
    }

    private void HtmlButtonClicked(string which)
    {
        HTMLMessage.Text = "HTML Button " + which + " Clicked";
    }
}

```

On attache trois événements à trois boutons de la page et on affiche quel bouton a été cliqué par la suite.

C# permet d'avoir des interactions bien plus fortes entre le *DOM* et *Silverlight*, mais il s'agit de concepts avancés qui ne seront pas abordés dans ce livre.

Pour plus d'informations, rendez-vous sur le site de MSDN (voir Annexe 3, *Webographie*).

4.5 Check-list

Nous avons étudié dans ce chapitre :

- ✓ l'ensemble des contrôles ASP.NET ;
- ✓ le contrôle media pour jouer des vidéos Silverlight ;
- ✓ le contrôle Silverlight pour mieux intégrer vos applications dans ASP.NET.

5



5.1 Le DataBinding en détails	214
5.2 Les Styles et ControlTemplates	220
5.3 Créer un UserControl	225
5.4 Les contrôles de la librairie System.Windows. Controls	246
5.5 Le contrôle DataGrid	253
5.6 Les contrôles Silverlight Toolkit de CodePlex	269
5.7 Check-list	271

Concepts avancés

Dans ce chapitre, nous allons approfondir les connaissances acquises au chapitre 2, *Le langage XAML*. Ce chapitre traitera principalement de l'interaction humain-machine.

Cette interaction peut être simplifiée :

- pour le développeur d'une part, grâce au *DataBinding* à deux sens, aux *Styles* et *ControlTemplates* et aux *UserControls* ;
- pour l'utilisateur de l'autre, grâce à une amélioration considérable de la qualité de l'interface.

5.1 Le DataBinding en détails

Jusqu'à présent, nous avons utilisé le *DataBinding* afin d'afficher des informations pour l'utilisateur. Cependant, le *DataBinding* permet aussi de demander à ce dernier des informations.

En effet, une fois liée à l'interface et pour peu que l'interface présente des éléments d'entrées utilisateur, une donnée peut être modifiée par l'utilisateur sans que le développeur n'ait à écrire la moindre ligne de code dans le code de la logique applicative.

L'exemple que nous allons développer ensemble va permettre à l'utilisateur de gérer une liste de films et de leur attribuer une appréciation en nombre d'étoiles.

Tout d'abord, il faut définir les objets Films. Un Film est un objet représenté par son *Titre*, son *Réalisateur* et son *Appréciation* :



Film.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace Filmothèque
{
    public class Film
    {
        private string titre;
        public string Titre
        {
            get { return titre; }
            set { titre = value; }
        }

        private string réalisateur;
        public string Réalisateur
        {
            get { return réalisateur; }
            set { réalisateur = value; }
        }
    }
}
```

```

        private int nombreDEtoiles;
        public int NombreDEtoiles
        {
            get { return nombreDEtoiles; }
            set { nombreDEtoiles = value; }
        }

        public Film()
        {
        }
    }
}

```

Au cours de ce chapitre, nous aurons aussi besoin d'une collection de films sur laquelle travailler. Prenons les devants et déclarons-la :



Collection de films

```

using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Collections.Generic;

namespace Filmotheque
{
    public static class CollectionDeFilms
    {
        public static List<Film> Films =
            new List<Film>()
            {
                new Film() { Titre="Retour vers le Présent 7",
                    Réalisateur="Takis Ergopoulos",
                    NombreDEtoiles=5},
                new Film() { Titre="Il faut sauver le magicien d'Oz",
                    Réalisateur="Orhan Benekhol",
                    NombreDEtoiles=1},
                new Film() { Titre="La guerre des Chtis",

```

```

        Réalisateur="Gautier Lebeaugros",
        NombreDEtoiles=4},
    new Film() { Titre="2009 l'odisée des Interfaces",
        Réalisateur="Ilkay Némètzakis",
        NombreDEtoiles=3},
    new Film() { Titre="Silverlight 2 Ze Movie",
        Réalisateur="Salvador Bargelot",
        NombreDEtoiles=4},
    };
}
}

```

DataContext

L'attribut `DataContext` est présent dans chaque élément d'interface *Silverlight*. C'est l'unité de ce qu'est la multitude `ItemSource` d'une `ListBox`.

Ainsi, lorsque vous "bindez" l'`ItemSource` d'une `ListBox` à une liste d'objets, par exemple une liste de films, chaque item généré comme enfant de cette `ListBox` par le `DataBinding` possèdera un `DataContext` référence du *Film* ayant servi de source à sa génération.

Pour simplifier, si nous lions par `DataBinding` la *CollectionDeFilms* à une `ListBox` :



Une ListBox (XAML)

```

<UserControl x:Class="Filmothèque.Page"
(...) >
    <Grid x:Name="LayoutRoot" Background="White">
        <ListBox Name="UneListBox"/>
    </Grid>
</UserControl>

```



Une ListBox (C#)

```

(using ...)

namespace Filmothèque
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
            UneListBox.ItemsSource = CollectionDeFilms.Films;
        }
    }
}

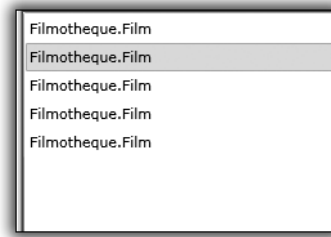
```



```

    }
}
}

```



► Fig. 5.1 :
Une ListBox liée à la Collection de films

Chaque élément `Filmothèque.Film` de la *ListBox* contient un *DataContext* pointant vers le film qu'il représente.

Dans le code de la logique applicative, il est alors possible de récupérer ce film à partir de l'élément *XAML*.

Qui plus est, un *DataContext* peut être rempli par le développeur sur d'autres éléments qu'un élément auto-généré par *DataBinding*.

Par exemple, avec cette interface :

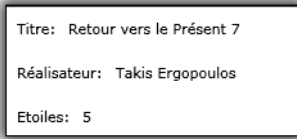
⚙ Interface recevant un DataContext codé en dure (Xaml)

```

<UserControl x:Class="Filmothèque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <StackPanel Orientation="Horizontal" Margin="4">
            <TextBlock Text="Titre:" Margin="5"/>
            <TextBlock Text="{Binding Path=Titre}" Margin="5"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal" Margin="4">
            <TextBlock Text="Réalisateur:" Margin="5"/>
            <TextBlock Text="{Binding Path=Réalisateur}" Margin="5"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal" Margin="4">
            <TextBlock Text="Etoiles:" Margin="5"/>
            <TextBlock Text="{Binding Path=NombreDEtoiles}" Margin="5"/>
        </StackPanel>
    </StackPanel>
</UserControl>

```

La ligne de code `LayoutRoot.DataContext = CollectionDeFilms.Films[0]`; donnera le résultat suivant :



► Fig. 5.2 :
Interface recevant un `DataContext` en dur

C'est exactement le même procédé qui est utilisé par la plateforme lorsqu'elle génère les items d'une *ListBox* à partir de *DataTemplate*. Pour chaque objet présent dans la collection de données en *ItemSource*, la plateforme ajoute aux enfants de la *ListBox* une copie du *DataTemplate* dont le *DataContext* est l'objet courant.

Interaction avec l'utilisateur

Comme il est écrit dans l'introduction de ce chapitre, le *DataBinding* ne sert pas seulement à afficher des données mais aussi à les modifier.

Si nous ajoutons à l'interface précédente quelques *TextBox*s liées par *binding* aux mêmes propriétés du film, et pour peu que la déclaration du *binding* le permette, le fait de modifier le texte contenu dans une *TextBox* changera directement les données en mémoire vive.

Pour permettre ce tour de magie, il faut spécifier à la déclaration du *binding* son attribut *Mode*.

Le *Mode* peut soit être :

- *OneWay* ; ce *Mode* est destiné uniquement à l'affichage de données.
- *TwoWay* ; ce *Mode* sert aussi bien à l'affichage et à la modification de données.

Le code *XAML* est-il le seul code à modifier pour ajouter cette fonctionnalité ? On aimerait bien mais en fait, si modifier uniquement le code *XAML* permet d'ajouter cette fonctionnalité, malheureusement, les modifications apportées aux données ne seront pas propagées à nouveau vers l'interface lors d'une modification. Dans un cas tel que celui-ci avec des éléments d'affichage de données, l'interface ne sera pas mise à jour.

Cela étant, ce cas reste isolé. Usuellement, il est inutile d'afficher des données dans deux éléments d'interface différents. Quoi que...

Pour pallier ce problème, il faut attraper un événement tel que *LostFocus* dans le code de la logique applicative et forcer la mise à jour du *DataContext*.

C'est ce que fait le code suivant :

DataBinding Modes + Mise à jour forcée (XAML)

```
<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <StackPanel Orientation="Horizontal" Margin="4">
            <TextBlock Text="Titre:" Margin="5"/>
            <TextBlock Text="{Binding Path=Titre}" Margin="5"/>
            <TextBox Text="{Binding Path=Titre, Mode=OneWay}" Margin="5"
                LostFocus="Input_LostFocus"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal" Margin="4">
            <TextBlock Text="Réalisateur:" Margin="5"/>
            <TextBlock Text="{Binding Path=Réalisateur}" Margin="5"/>
            <TextBox Text="{Binding Path=Réalisateur, Mode=TwoWay}"
                Margin="5"
                LostFocus="Input_LostFocus"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal" Margin="4">
            <TextBlock Text="Etoiles:" Margin="5"/>
            <TextBlock Text="{Binding Path=NombreDEtoiles}" Margin="5"/>
            <Slider Value="{Binding Path=NombreDEtoiles, Mode=TwoWay}"
                Minimum="0" Maximum="5"
                Width="100" Margin="5"
                LostFocus="Input_LostFocus"/>
        </StackPanel>
    </StackPanel>
</UserControl>
```

DataBinding Modes + Mise à jour forcée (C#)

```
using (...);

namespace Filmotheque
{
    public partial class Page : UserControl
    {
        private bool isIntialized = false;

        public Page()
        {
            InitializeComponent();
        }
    }
}
```

```

        isIntialized = true;
        LayoutRoot.DataContext = CollectionDeFilms.Films[0];
    }

    private void Input_LostFocus(object sender, RoutedEventArgs e)
    {
        ResetDataContext();
    }

    private void ResetDataContext()
    {
        if (!isIntialized) return;
        LayoutRoot.DataContext = null;
        LayoutRoot.DataContext = CollectionDeFilms.Films[0];
    }
}

```

5.2 Les Styles et ControlTemplates

Style

Lors de la création d'une interface *XAML*, de nombreux attributs vous aident à configurer l'affichage des différents éléments d'interface.

Qu'il s'agisse des attributs *Background*, *Foreground*, *StrokeThickness*, *Fill* ou d'autres attributs, ils se retrouvent spontanément dans chacun de vos éléments d'interface.

Dans le but d'obtenir une interface cohérente, vous serez amené à copier ces éléments dits de style d'un contrôle utilisateur à l'autre.

Pour éviter une prolifération ingérable de code similaire à travers votre application, l'utilisation de styles s'impose. Un style est un ensemble d'attributs prédéfinis applicables à un type d'élément pour en configurer le visuel.

Considérons un code lourd tel que celui-ci :



Code lourd

```

<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="100" Height="80">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <TextBlock Foreground="Black" FontSize="15" FontFamily="Arial"

```

```

        FontWeight="Bold" HorizontalAlignment="Center"
        Text="TextBlock1"/>
    <TextBlock Foreground="Black" FontSize="15" FontFamily="Arial"
        FontWeight="Bold" HorizontalAlignment="Center"
        Text="TextBlock2"/>
    <TextBlock Foreground="Black" FontSize="15" FontFamily="Arial"
        FontWeight="Bold" HorizontalAlignment="Center"
        Text="TextBlock3"/>
    <TextBlock Foreground="Black" FontSize="15" FontFamily="Arial"
        FontWeight="Bold" HorizontalAlignment="Center"
        Text="TextBlock4"/>

</StackPanel>
</UserControl>

```

Remplaçons la répétition d'attributs de style par un élément Style unique :

Code allégé par Style

```

<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="100" Height="80">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <StackPanel.Resources>
            <Style x:Key="BoldTextBlockStyle" TargetType="TextBlock">
                <Setter Property="Foreground" Value="Black"/>
                <Setter Property="FontSize" Value="15"/>
                <Setter Property="FontFamily" Value="Arial"/>
                <Setter Property="FontWeight" Value="Bold"/>
                <Setter Property="HorizontalAlignment" Value="Center"/>
            </Style>
        </StackPanel.Resources>

        <TextBlock Style="{StaticResource BoldTextBlockStyle}"
            Text="TextBlock1"/>
        <TextBlock Style="{StaticResource BoldTextBlockStyle}"
            Text="TextBlock2"/>
        <TextBlock Style="{StaticResource BoldTextBlockStyle}"
            Text="TextBlock3"/>
        <TextBlock Style="{StaticResource BoldTextBlockStyle}"
            Text="TextBlock4"/>

    </StackPanel>
</UserControl>

```

Le code devient bien plus lisible et on peut en assurer facilement la maintenance. Changer globalement l'aspect des *TextBlock* du *Style* *BoldTextBlock* se fait en une modification au lieu de quatre.

Le résultat, quant à lui, reste le même :



► Fig. 5.3 :
TextBlock avec Style

Un style possède comme attributs une *x:Key* le représentant et un *TargetType*. Le *TargetType* est le type d'élément *XAML* auquel le style peut être appliqué. Laisser vide cette dernière propriété vous permettra d'appliquer ce style à n'importe quel élément *XAML*. En contrepartie, il est dangereux de s'y résoudre par simplicité vu que certains attributs (tels que *Foreground*) ne sont pas partagés par tous les éléments *XAML*.

Les éléments enfants des styles sont les *setters*. Un setter configure l'attribut du nom de *Property* avec la valeur *Value*.

ControlTemplate

Un *ControlTemplate* est une sorte de style qui, au lieu de configurer certains attributs d'un élément d'interface, remplace complètement son affichage par un arbre *XAML* :



Exemple de ControlTemplate

```
<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="150" Height="150">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <StackPanel.Resources>
            <Style x:Key="BoldTextBlockStyle" TargetType="TextBlock">
                <Setter Property="Foreground" Value="Black"/>
                <Setter Property="FontSize" Value="15"/>
                <Setter Property="FontFamily" Value="Arial"/>
                <Setter Property="FontWeight" Value="Bold"/>
                <Setter Property="HorizontalAlignment" Value="Center"/>
            </Style>

            <ControlTemplate x:Key="ActionControlTemplate"
                TargetType="Button">
                <Border Background="Silver"
```

```

        BorderThickness="2"
        BorderBrush="Blue"
        CornerRadius="0,0,15,0"
        Margin="5">
    <StackPanel Orientation="Vertical" Margin="5">
        <TextBlock
            Style="{StaticResource BoldTextBlockStyle}"
            Text="{TemplateBinding Content}"
            HorizontalAlignment="Left"/>
        <Button Click="{TemplateBinding Click}">
            <TextBlock
                Style="{StaticResource BoldTextBlockStyle}"
                HorizontalAlignment="Right"
                Text="Do it!"/>
        </Button>
    </StackPanel>
</Border>
</ControlTemplate>
</StackPanel.Resources>

<Button Template="{StaticResource ActionControlTemplate}"
        Content="Une Action"/>

<Button Template="{StaticResource ActionControlTemplate}"
        Content="Autre Action"/>
</StackPanel>
</UserControl>

```



► Fig. 5.4 :
Exemple de ControlTemplate

Étrangement, assigner un *ControlTemplate* à un élément *XAML* ne se fait pas par l'attribut *ControlTemplate* mais via l'attribut *Template*.

À l'intérieur même du *ControlTemplate*, un nouveau type de *binding* montre son nez : le *TemplateBinding*.

Un *TemplateBinding* permet à un *ControlTemplate* de récupérer la valeur d'un des attributs de l'élément qu'il utilise.

Ainsi en écrivant `Text="{TemplateBinding Content}"`, nous allons rechercher le `Content` des deux boutons qui utilisent `ActionControlTemplate`.

Ceci est dangereux. En effet, nous avons assigné par `TemplateBinding` à l'attribut `Text` d'une *TextBox* une valeur qui peut ne pas être du texte. Rappelons-nous qu'un *Button* peut contenir n'importe quel autre élément *XAML*, ce qui n'est pas le cas de l'attribut `Text`, qui lui, accepte uniquement du texte.

Pour pallier ce risque, il est préférable d'utiliser un `ContentPresenter`. Un `ContentPresenter` est un élément *XAML* qui possède la capacité d'afficher son contenu.

Modifions le code du `ControlTemplate` :



ControlTemplate avec ContentTemplate

```
<UserControl x:Class="Filmotheque.Page"
    (...)

    <ControlTemplate x:Key="ActionControlTemplate"
        TargetType="Button">
        <Border Background="Silver"
            BorderThickness="2"
            BorderBrush="Blue"
            CornerRadius="0,0,15,0"
            Margin="5">
            <StackPanel Orientation="Vertical"
                Margin="5">
                <ContentPresenter
                    Content="{TemplateBinding Content}"
                    HorizontalAlignment="Left"/>
                <Button Click="{TemplateBinding Click}">
                    <TextBlock
                        Style="{StaticResource BoldTextBlockStyle}"
                        HorizontalAlignment="Right"
                        Text="Do it!"/>
                </Button>
            </StackPanel>
        </Border>
    </ControlTemplate>
</StackPanel.Resources>

<Button Template="{StaticResource ActionControlTemplate}"
    Content="Une Action"/>

<Button Template="{StaticResource ActionControlTemplate}">
    <StackPanel>
```



```

        <TextBlock Text="Ecrire un livre"/>
        <TextBlock Text="Publier un livre"/>
    </StackPanel>
</Button>
</StackPanel>
</UserControl>

```



► Fig. 5.5 :
ControlTemplate avec ContentTemplate

Dans ce nouvel exemple, que le contenu du bouton soit un simple texte ou un arbre XAML complexe, le résultat est le même.

5.3 Créer un UserControl

Toutes ces configurations nous conduisent vers le concept d'*UserControl*. Un *UserControl*, ou contrôle utilisateur en français, est un élément d'interface XAML défini par le développeur.

Il est écrit à l'aide d'un fichier XAML séparé du corps de l'application mais aussi d'un fichier de code applicatif personnel.

En effet, un contrôle utilisateur n'est plus seulement un élément de configuration visuelle, mais aussi un élément englobant une part de logique interne.

Pour créer un *UserControl*, il faut :

- ajouter ses codes XAML et C# à la solution *Silverlight* ;
- définir l'interface XAML ;
- écrire la logique applicative.

Pour utiliser un *UserControl*, il faut :

- Si le *UserControl* est défini dans une autre assembly, s'assurer qu'elle a été compilée pour *Silverlight* et en ajouter la référence au projet *Silverlight*.
- Ajouter à l'application l'espace de noms (namespace) contenant l'*UserControl*.

UserControl ClickMe

Pour appréhender facilement ce concept de contrôle utilisateur, nous allons en écrire un basique. Cet *UserControl* contiendra un bouton et changera le contenu de ce bouton lorsqu'on cliquera dessus.

Voici l'interface de ClickMe :



ClickMe.xaml

```
<UserControl x:Class="Filmotheque.ClickMe"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button Name="ClickMeButton"
            Content="Click Me!"
            Click="Button_Click"/>
    </Grid>
</UserControl>
```

Voici le code de la logique applicative :



ClickMe.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace Filmotheque
{
    public partial class ClickMe : UserControl
    {
        public ClickMe()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
```

```

    {
        ClickMeButton.Content = "Merci";
    }
}

```

Utilisation de l'UserControl ClickMe :

Utilisation du UserControl ClickMe

```

<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:Filmotheque"
    Width="150" Height="150">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <MyApp:ClickMe Margin="5"/>
        <MyApp:ClickMe Margin="5"/>
        <MyApp:ClickMe Margin="5"/>
        <MyApp:ClickMe Margin="5"/>
    </StackPanel>
</UserControl>

```



► Fig. 5.6 :
UserControl ClickMe sous différents états

UserControl Ranking

Dans l'optique de l'application de gestion de films, un *UserControl* permettant d'attribuer un nombre d'étoiles à un film de manière visuel peut avoir son importance.

Le résultat final auquel nous aimerions arriver est le suivant :



► Fig. 5.7 :
Ranking UserControl

Il doit être possible d'écrire le code suivant :



Objectif code XAML Ranking UserControl

```
<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:Filmotheque"
    Width="400" Height="300">
    <StackPanel x:Name="LayoutRoot" Background="White">
        <MyApp:Ranking NombreDEtoile="3"/>
        <MyApp:Ranking
            NombreDEtoile="{Binding Path=NombreDEtoile, Mode=TwoWay}"/>
    </StackPanel>
</UserControl>
```

Un point reste à voir avant d'arriver à ce résultat : Comment ajouter à un *UserControl* une propriété pouvant servir d'attributs dans le code *XAML*.

Les DependencyProperty

Une *DependencyProperty* est une propriété déclarée dans le code applicatif d'un *UserControl* grâce à la méthode *Register* de la classe *DependencyProperty*.

C'est une propriété qui au lieu d'être enregistrée dans une variable privée du code applicatif, est enregistrée par le Framework lui-même d'une manière qui importe peu.

Quoi qu'il en soit, seuls les *DependencyProperties* sont stables lorsque vous désirez utiliser un attribut dans le code *XAML*.

L'utilisation d'une propriété publique du code applicatif dans ce but peut avoir des effets malencontreux et imprévisibles.

L'emploi d'une *DependencyProperty* à la place d'une propriété publique active la gestion des *Animations*, des *Styles*, des *Templates* et du *binding* sur cette propriété.

Déclaration d'une *DependencyProperty* :

```
public int MyProperty
{
    get { return (int)GetValue(MyPropertyProperty); }
    set { SetValue(MyPropertyProperty, value); }
}
```

Déclaration d'une DependencyProperty

```
public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.Register("MyProperty",
                                typeof(int),
                                typeof(ownerclass),
                                new PropertyMetadata(0));
```

La fonction `DependencyProperty.Register` demande 4 paramètres :

- le nom de la propriété ;
- son type ;
- le type de sa classe mère, la classe qui ôte cette `DependencyProperty` (généralement la classe dans laquelle cette propriété est déclarée) ;
- une instance de la classe `PropertyMetadata` spécifiant la valeur initiale de la propriété.

Dans ce cas, nous déclarons donc :

- une propriété du nom `MyProperty` ;
- de type nombre entier (`int`) ;
- dont la classe hôte est `ownerclass` ;
- de valeur initiale 0.

Création de l'UserControl Ranking

Rappel du cahier des charges : Créer un *UserControl* affichant de 1 à 5 étoiles selon une propriété `NombreDEtoiles`.

Pour ce faire, nous allons créer deux contrôles utilisateur :

- Le premier du nom de `Star` étant une étoile unique, capable de varier de couleur (jaune ou gris) quand on clique dessus ou lors de l'appel d'une de ses fonctions publiques.
- Le deuxième du nom de `Ranking` sera un conteneur de cinq *UserControl* `Star`, coordonnant leurs travaux.

Star

Pour créer l'*UserControl* `Star`, nous allons partir d'une `CheckBox`. En effet, comme nous l'avons vu au chapitre 2, *Le langage XAML*, une `CheckBox` possède un attribut `IsChecked` à deux états. Cet attribut nous sera utile pour savoir si l'étoile est jaune (`IsChecked=True`) ou si l'étoile est grise (`IsChecked=False`).

Nous allons ensuite en redéfinir le *ControlTemplate* pour lui donner l'apparence d'une étoile.

Dans le code suivant, vous découvrirez l'utilisation de l'élément géométrique *Path*. Cet élément prend comme valeur de l'attribut *Data* une série de points représentés par des coordonnées géométriques.

L'attribut *Fill* de ce *Path* est lié par *TemplateBinding* au *Background* de la *CheckBox*.

C'est en faisant varier ce *Background* de *YellowStarBrush* à *GreyStarBrush* dans le code applicatif que nous changerons le visuel de l'étoile :



Star.xaml

```
<UserControl x:Class="Filmotheque.Star"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.Resources>
            <SolidColorBrush x:Key="YellowStarBrush" Color="#FFFF00"/>
            <SolidColorBrush x:Key="GreyStarBrush" Color="#C0C0C0"/>

            <Style x:Key="EmptyStarStyle" TargetType="CheckBox">
                <Setter Property="Height" Value="40"/>
                <Setter Property="Width" Value="40"/>
                <Setter Property="Margin" Value="2"/>
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate x:Name="EmptyStarTemplate">
                            <Canvas>
                                <Canvas Canvas.Left="5"
                                    Canvas.Top="15">
                                    <Path Stroke="#000080"
                                        Fill="{TemplateBinding Background}"
                                        MouseLeftButtonDown="Path_MouseLeftButtonDown"
                                        MouseEnter="Path_MouseEnter"
                                        MouseLeave="Path_MouseLeave"
                                        StrokeThickness="3"
                                        StrokeStartLineCap="Round"
                                        StrokeEndLineCap="Round"
                                        StrokeLineJoin="Round"
                                        Data="M 0,0 1 10,0 1 5,-10
                                            1 5,10 1 10,0 1 -7,10 1 2,10
                                            1 -10,-5 1 -10,5 1 2,-10 Z"/>
                                </Canvas>
                            </Canvas>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </Style>
        </Grid.Resources>
    </Grid>
</UserControl>
```

```

        </Setter.Value>
    </Setter>
</Style>
</Grid.Resources>

    <CheckBox Name="Star1"
        IsChecked="True"
        Style="{StaticResource EmptyStarStyle}"
        Background="{StaticResource YellowStarBrush}"/>

</Grid>
</UserControl>

```

Trois événements sont utilisés dans ce *ControlTemplate* :



► Fig. 5.8 :
Etoile solitaire

- **MouseDown** lorsque l'utilisateur cliquera sur l'étoile, son attribut **IsChecked** sera inversé.
- **MouseEnter** et **MouseLeave** lorsque la souris de l'utilisateur se trouve au-dessus de l'étoile, nous en changerons le **Background** en Orange.

⚙ Star.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace Filmotheque
{
    public partial class Star : UserControl
    {
        public event EventHandler OnColorChanged;

        public bool? IsChecked
        {
            get { return (bool?)GetValue(IsCheckedProperty); }
        }
    }
}

```

```

        set
        {
            SetValue(IsCheckedProperty, value);
            Star1.IsChecked = value;
            SetBackground();
        }
    }

    public static readonly DependencyProperty
        IsCheckedProperty = DependencyProperty.Register(
            "IsChecked",
            typeof(bool?),
            typeof(Star),
            new PropertyMetadata(true));

    public Star()
    {
        InitializeComponent();
        SetBackground();
    }

    private void Path_MouseLeftButtonDown(object sender,
        MouseButtonEventArgs e)
    {
        Star1.IsChecked = !Star1.IsChecked;
        SetBackground();
        if (OnColorChanged != null)
            OnColorChanged(this, EventArgs.Empty);
    }

    private void SetBackground()
    {
        if (Star1.IsChecked.HasValue && Star1.IsChecked.Value)
            Star1.Background = (SolidColorBrush)
                LayoutRoot.Resources["YellowStarBrush"];
        else
            Star1.Background = (SolidColorBrush)
                LayoutRoot.Resources["GreyStarBrush"];
    }

    #region MouseOverManagement
    private Brush SolideStateBackground;
    private void Path_MouseEnter(object sender, MouseEventArgs e)
    {
        SolideStateBackground = Star1.Background;
        Star1.Background = new SolidColorBrush(Colors.Orange);
    }

```



```

    }

    private void Path_MouseLeave(object sender, MouseEventArgs e)
    {
        Star1.Background = SolideStateBackground;
    }
    #endregion MouseOverManagement
}
}

```

Un point est à mettre en évidence dans ce code applicatif : l'événement `OnColorChanged` qui préviendra l'*UserControl* Ranking que l'utilisateur a cliqué sur l'étoile.

Ranking

Interface de l'*UserControl* Ranking :

 Ranking.xaml

```

<UserControl x:Class="Filmotheque.Ranking"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:Filmotheque"
    Width="Auto" Height="Auto" Margin="2" >
    <Grid x:Name="LayoutRoot" Background="White">

        <Border HorizontalAlignment="Center"
            VerticalAlignment="Center"
            Background="White" CornerRadius="5"
            BorderBrush="#000080" BorderThickness="4">
            <StackPanel Name="RankingPanel" Orientation="Horizontal">
                <MyApp:Star Name="Star1"/>
                <MyApp:Star Name="Star2"/>
                <MyApp:Star Name="Star3"/>
                <MyApp:Star Name="Star4"/>
                <MyApp:Star Name="Star5"/>
            </StackPanel>
        </Border>
    </Grid>
</UserControl>

```

Comme prévu, du côté de l'interface du contrôle utilisateur Ranking, il s'agit seulement d'une collection de contrôle utilisateur Star.



► Fig. 5.9 :
Ranking

On comprend ici l'utilité d'avoir créé un contrôle utilisateur pour les étoiles ; il est devenu très aisé de faire varier le nombre d'étoiles qu'un *Ranking* possède. Qui plus est, pour aller plus loin, il est possible de faire de ce nombre d'étoiles maximal une *DependencyProperty* configurable.

Le code applicatif de *Ranking* va devoir rester à l'écoute des clics de l'utilisateur sur chaque étoile et stocker en mémoire le nombre d'étoiles qui lui est attribué.

En effet, lorsque l'utilisateur clique sur l'étoile 3, toutes les étoiles précédentes doivent passer à l'état jaune et inversement, toutes les étoiles suivantes doivent passer à l'état gris.



Ranking.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace Filmothèque
{
    public partial class Ranking : UserControl
    {
        public int NombreDEtoiles
        {
            get { return (int)GetValue(NombreDEtoilesProperty); }
            set
            {
                SetValue(NombreDEtoilesProperty, value);
                SetRanking(value);
            }
        }

        public static readonly DependencyProperty
            NombreDEtoilesProperty =
```

```

        DependencyProperty.Register("NombreDEtoiles",
                                     typeof(int),
                                     typeof(Ranking),
                                     new PropertyMetadata(0));

public Ranking()
{
    InitializeComponent();
    foreach(Star star in RankingPanel.Children)
        star.OnColorChanged +=
            new EventHandler(star_OnColorChanged);
}

void star_OnColorChanged(object sender, EventArgs e)
{
    Star star = (sender as Star);
    int Position = int.Parse(star.Name.Replace("Star", ""));
    SetRanking(Position);
}

public void SetRanking(int NombreDEtoiles)
{
    for (int i = 1; i <= 5; i++)
    {
        Star star =
            (RankingPanel.FindName("Star" + i) as Star);
        star.IsChecked = (i <= NombreDEtoiles);
    }
}
}

```

Intégration du contrôle utilisateur Ranking dans une application Silverlight

Il suffit de reprendre le code étudié il y a quelque pages dans les spécifications de l'*UserControl* Ranking et d'y ajouter un *DataContext* en code applicatif pour vérifier qu'il fonctionne correctement :



Objectif code XAML Ranking UserControl

```

<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

```

```

xmlns:MyApp="clr-namespace:Filmotheque"
Width="400" Height="300">
<StackPanel x:Name="LayoutRoot" Background="White">
    <MyApp:Ranking NombreDEtoile="3"/>
    <MyApp:Ranking
        NombreDEtoile="{Binding Path=NombreDEtoile, Mode=TwoWay}"/>
</StackPanel>
</UserControl>

```



Code applicatif

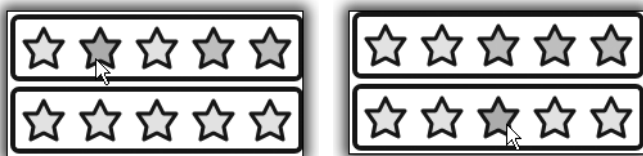
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

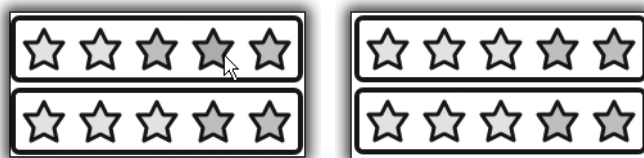
namespace Filmotheque
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
            LayoutRoot.DataContext = CollectionDeFilms.Films[0];
        }
    }
}

```

Le résultat est à la hauteur de nos espérances :



► Fig. 5.10 : Ranking en action



► Fig. 5.11 : Ranking en action

MediaElement

Dans une optique totalement différente, nous allons maintenant nous pencher sur un contrôle *Silverlight* des plus puissants : le contrôle `MediaElement`.

Le contrôle `MediaElement` permet d'afficher et d'interagir avec une vidéo à l'intérieur d'une application *Silverlight*.

Attribut `MediaElement.Source` en XAML bug

Lorsque vous ajoutez un contrôle `MediaElement` à votre application *Silverlight*, n'en configurez pas la source à partir du code *XAML*, vous recevrez dans ce cas une erreur de chargement de la vidéo (*Erreur 4001*). Ceci est un bogue ouvert chez *Microsoft* mais restons calme, ce bogue n'est pas bloquant, il suffit de configurer la source des `MediaElement` à partir du code applicatif.

Interface d'un exemple d'utilisation du contrôle `MediaElement` :

Exemple de `MediaElement`

```
<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:Filmotheque"
    Width="430" Height="300">
    <StackPanel x:Name="LayoutRoot" Background="Black">
        <StackPanel
            Orientation="Horizontal">
            <StackPanel HorizontalAlignment="Center"
                VerticalAlignment="Center">

                <Button Content="Play" Margin="5"
                    Click="Play_Click"/>
                <Button Content="Pause" Margin="5"
                    Click="Pause_Click"/>
                <Button Content="Stop" Margin="5"
                    Click="Stop_Click"/>
            </StackPanel>
        </StackPanel>
    </StackPanel>
</UserControl>
```

```

        <TextBlock Foreground="White"
                  VerticalAlignment="Center"
                  Margin="5" >Volume</TextBlock>
        <Slider Name="VolumeSlider"
                VerticalAlignment="Center"
                Minimum="0" Maximum="1"
                Value="0.5" Width="70"
                ValueChanged="VolumeSlider_ValueChanged"/>

        <ToggleButton Name="MuteToggleButton"
                      Content="Mute"
                      Margin="5" IsChecked="False"
                      Checked="Mute"
                      Unchecked="UnMute"/>

    </StackPanel>

    <Border Background="Black"
            BorderBrush="WhiteSmoke"
            BorderThickness="3"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            Margin="5"
            CornerRadius="5">
        <MediaElement Name="myMediaElement"
                      Width="320"
                      Height="240"
                      Margin="5"
                      Stretch="Fill"
                      AutoPlay="False"
                      MediaFailed="MediaElement_MediaFailed"/>
    </Border>

    </StackPanel>
</Grid>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*/>
    <ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<TextBlock Foreground="White" Margin="5"
            Grid.Column="0"
            VerticalAlignment="Center">Seek To</TextBlock>
<Slider Name="TimeLineSlider" Margin="5"
        Minimum="0"
        Grid.Column="1"/>
<TextBlock Name="TimeLineTextBlock" Grid.Column="2"

```

```

Text="00:00:00/00:00:00"
Foreground="White" Margin="5"/>
</Grid>
</StackPanel>
</UserControl>

```



► Fig. 5.12 :
Exemple de
MediaElement

Dans cette application *Silverlight*, vous retrouvez un `MediaElement` et quelques autres contrôles destinés à interagir avec lui.

Les différents contrôles ajoutés parlent d'eux-mêmes, le bouton **Play** sert à démarrer la vidéo, le bouton **Pause** à l'arrêter temporairement, etc.

Le code applicatif, quant à lui, cache quelques subtilités :



Code applicatif de l'exemple de `MediaElement`

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Data;
using System.Threading;

namespace Filmotheque
{

```

```

public partial class Page : UserControl
{
    public Page()
    {
        InitializeComponent();
        myMediaElement.Source =
            new Uri(@"media/Lake.wmv", UriKind.Relative);
    }

    private void MediaElement_MediaFailed(object sender,
        ExceptionRoutedEventArgs e)
    {
    }

    private void Play_Click(object sender, RoutedEventArgs e)
    {
        TimeLineSlider.Maximum =
            myMediaElement.NaturalDuration.TimeSpan.TotalSeconds;
        TimeLineSlider.Value =
            myMediaElement.Position.TotalSeconds;

        myMediaElement.Play();

        Timer T = new Timer(new TimerCallback(delegate
        {
            Dispatcher.BeginInvoke(new Action(delegate
            {
                TimeLineSlider.SetValue(Slider.ValueProperty,
                    myMediaElement.Position.TotalSeconds);
                TimeLineTextBlock.Text =
                    myMediaElement.Position.Hours + ":" +
                    myMediaElement.Position.Minutes + ":" +
                    myMediaElement.Position.Seconds + "/" +
                    myMediaElement.NaturalDuration.TimeSpan.Hours
                    + ":" +
                    myMediaElement.NaturalDuration.TimeSpan.Minutes
                    + ":" +
                    myMediaElement.NaturalDuration.TimeSpan.Seconds;
            }));
        }));
        T.Change(0, 1000);
    }
}

```



```

private void Pause_Click(object sender, RoutedEventArgs e)
{
    myMediaElement.Pause();
}

private void Stop_Click(object sender, RoutedEventArgs e)
{
    myMediaElement.Stop();
}

private void VolumeSlider_ValueChanged(object sender,
    RoutedPropertyChangedEventArgs<double> e)
{
    if (VolumeSlider != null && myMediaElement != null)
        myMediaElement.Volume = VolumeSlider.Value;
}

private void Mute(object sender, RoutedEventArgs e)
{
    myMediaElement.IsMuted = true;
}

private void UnMute(object sender, RoutedEventArgs e)
{
    myMediaElement.IsMuted = false;
}

}

```

Pour pallier le bogue relatif à l'attribut Source des MediaElement, c'est dans ce code applicatif que l'on se doit de l'assigner.



► Fig. 5.13 :
Exemple de
MediaElement en
fonctionnement

Pour faire varier l'attribut Valeur du *Silder TimeLineSlider*, nous avons utilisé un objet de la *plateforme .Net* du nom de Dispatcher.

Dispatcher

Dans une application *Silverlight*, comme dans une application *WPF*, l'interface est propriété d'un unique *thread*. C'est dans ce *thread* que s'exécutent les différents événements.

Cependant, il est parfois nécessaire d'attaquer l'interface à partir d'un autre *thread*. C'est impossible.

Le Dispatcher est un objet gérant l'ordre d'appel des fonctions du *thread* de l'interface. Tout ce qu'il est possible de faire à partir d'un autre *thread* est de demander au Dispatcher de mettre en queue l'appel d'une fonction.

Quand le Dispatcher en trouvera le temps, il l'exécutera.

Cette demande se fait à partir de la méthode *BeginInvoke* du Dispatcher :



Utilisation du Dispatcher

```
Thread autre que le thread interface
{
    /* interaction code interface impossible */
    Dispatcher.BeginInvoke(new Action(delegate
    {
        /* interaction code interface possible */
    }));
}
```

Passer en mode Plein écran

Une fonctionnalité toujours appréciée par un utilisateur lorsqu'il regarde une vidéo est la possibilité de passer en mode Plein écran.

Silverlight permet cette fonctionnalité grâce à la propriété bien cachée : *App.Current.Host.Content.IsFullScreen*.

Malheureusement dans notre cas, en assignant cette propriété à *True*, c'est l'intégralité de l'application *Silverlight* qui passe en mode Plein écran, alors que nous voudrions voir uniquement la vidéo.

Pour compléter cette fonctionnalité, il faut réorganiser la structure de l'application.

L'objet `App.Current.Host.Content` contient deux autres propriétés qui vont nous servir :

- `ActualHeight` est la hauteur de l'écran de l'utilisateur.
- `ActualWidth` est la largeur de l'écran de l'utilisateur.

Cet objet contient aussi un événement : `FullScreenChanged` sur lequel nous ajouterons une méthode réorganisant la structure de l'application.

Ajout d'un bouton **FullScreen** à l'interface et modification du code XAML :



MediaElement FullScreen (Xaml)

```
<UserControl x:Class="Filmotheque.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyApp="clr-namespace:Filmotheque"
    Width="Auto" Height="Auto">
    <StackPanel x:Name="LayoutRoot" Background="Black">
        <StackPanel
            Orientation="Horizontal">
            <StackPanel HorizontalAlignment="Center"
                Name="ButtonStackPanel"
                VerticalAlignment="Center">

                <Button Content="Play" Margin="5" Click="Play_Click"/>
                <Button Content="Pause" Margin="5" Click="Pause_Click"/>
                <Button Content="Stop" Margin="5" Click="Stop_Click"/>
                <Button Content="FullScreen" Margin="5"
                    Click="FullScreen_Click"/>

            <TextBlock Foreground="White" VerticalAlignment="Center"
                Margin="5" >Volume</TextBlock>
            <Slider Name="VolumeSlider" VerticalAlignment="Center"
                Minimum="0" Maximum="1" Value="0.5" Width="70"
                ValueChanged="VolumeSlider_ValueChanged"/>

            <ToggleButton Name="MuteToggleButton" Content="Mute"
                Margin="5" IsChecked="False"
                Checked="Mute"
                Unchecked="UnMute"/>

        </StackPanel>

    <Border Name="EcranBorder"
        Background="Black"
        BorderBrush="WhiteSmoke"
        BorderThickness="3"
```

```

        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Margin="5"
        Width="330"
        Height="250"
        CornerRadius="5">
        <MediaElement Name="myMediaElement"
            Margin="5"
            Stretch="Fill"
            AutoPlay="False"
            MediaFailed="MediaElement_MediaFailed"/>
    </Border>

</StackPanel>
<Grid Name="SliderGrid" Width="400"
    HorizontalAlignment="Left">
</StackPanel>
</UserControl>

```

Lors de l'événement Click du bouton **FullScreen**, nous allons :

- Assigner les attributs Visible des éléments ButtonStackPanel et SliderGrid à Collapsed. La valeur Collapsed de cet attribut signifie que les éléments ne doivent ni être affichés sur l'interface, ni occuper la moindre place résiduelle dans l'interface.
- Redimensionner le Border EcranBorder pour qu'il occupe toute la place disponible sur l'écran de l'utilisateur.

Modification du code applicatif :



MediaElement FullScreen

```

(...)
public Page()
{
    InitializeComponent();
    myMediaElement.Source = new Uri(@"media/Lake.wmv",
    ➔ UriKind.Relative);

    App.Current.Host.Content.FullScreenChanged +=
        new EventHandler(Content_FullScreenChanged);
}

void Content_FullScreenChanged(object sender, EventArgs e)
{
    if (!App.Current.Host.Content.IsFullScreen)

```

```

    {
        ButtonStackPanel.Visibility = Visibility.Visible;
        SliderGrid.Visibility = Visibility.Visible;
        EcranBorder.SetValue(Border.HeightProperty,
            (double)250);
        EcranBorder.SetValue(Border.WidthProperty,
            (double)330);
    }
    else
    {
        ButtonStackPanel.Visibility = Visibility.Collapsed;
        SliderGrid.Visibility = Visibility.Collapsed;
        EcranBorder.SetValue(Border.HeightProperty,
            App.Current.Host.Content.ActualHeight - 10);
        EcranBorder.SetValue(Border.WidthProperty,
            App.Current.Host.Content.ActualWidth - 10);
    }
}

private void FullScreen_Click(object sender, RoutedEventArgs e)
{
    App.Current.Host.Content.IsFullScreen = true;
}

```



► Fig. 5.14 :
MediaElement modifié
pour supporter le
FullScreen



► Fig. 5.15 : MediaElement en mode FullScreen

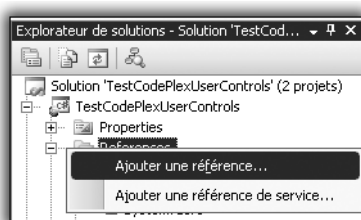
5.4 Les contrôles de la librairie System.Windows.Controls

En plus des contrôles utilisateur *Silverlight* de base, la *plateforme .Net* permet l'utilisation de 6 contrôles utilisateur supplémentaires.

Cinq de ces contrôles sont présents dans la bibliothèque *System.Windows.Controls*.

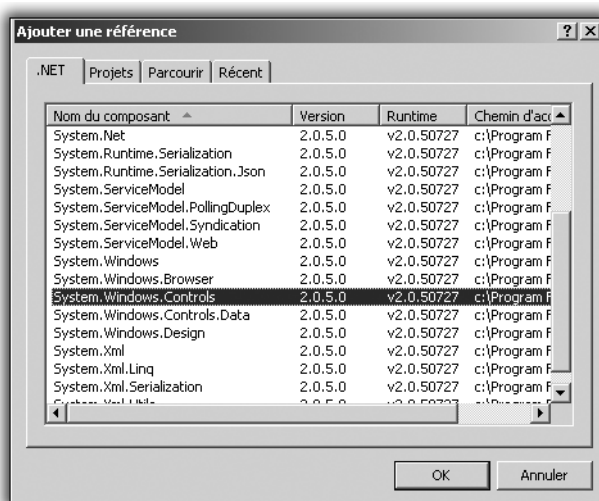
Pour utiliser cette librairie, vous devez l'ajouter en référence dans votre projet *Silverlight* :

- 1 Cliquez du bouton droit sur **Reference** dans l'Explorateur de solution.
- 2 Sélectionnez l'action **Ajouter une référence**.



► Fig. 5.16 :
Add Reference

- 3 Dans la boîte de dialogue **Ajouter une référence**, sélectionnez l'onglet **.Net**.
- 4 Sélectionnez **System.Windows.Controls**.



► Fig. 5.17 :
Boîte de dialogue
Ajouter une référence

- 5 Cliquez sur OK

Calendar

Le premier de ces contrôles utilisateur est un calendrier :

Exemple de Calendar

```
<UserControl x:Class="TestWindowsControls.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:controls="clr-namespace:System.Windows.Controls;
  assembly=System.Windows.Controls"
```

```

xmlns:data="clr-namespace:System.Windows.Controls;
➤ assembly=System.Windows.Controls.Data"
Width="Auto" Height="Auto">
<Grid x:Name="LayoutRoot" Background="White">
    <controls:Calendar Name="calendar"/>
</Grid>
</UserControl>

```



► Fig. 5.18 :
Exemple de Calendar

Les attributs permettant de configurer ce calendrier sont :

- DisplayDate spécifie la date à afficher par défaut.
- DisplayMode peut prendre les valeurs :
 - Decade affiche 10 années à l'utilisateur.
 - Year affiche 12 mois à l'utilisateur.
 - Mounth affiche un mois à l'utilisateur sous forme de tableau de jours.
- DisplayDateStart et DisplayDateEnd permettent d'afficher seulement une certaine période de temps sur le calendrier.

Par exemple, pour afficher uniquement les 4 jours de chaque côté du 29 juillet 1985, il faut modifier le code applicatif de l'application :



Exemple de Calendar 2

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

```



```

using Filmotheque;

namespace TestWindowsControls
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();

            DateTime Lundi29Juillet85 = new DateTime(1985, 7, 29);
            calendar.DisplayDate = Lundi29Juillet85;
            calendar.DisplayDateStart = Lundi29Juillet85.AddDays(-4);
            calendar.DisplayDateEnd = Lundi29Juillet85.AddDays(4);
        }
    }
}

```



► Fig. 5.19 :
Calendar de 8 jours

DatePicker

Le DatePicker fonctionne majoritairement de la même façon que le Calendar.

Il accepte lui aussi les attributs DisplayDate, DisplayDateStart et DisplayDateEnd. L'attribut DisplayMode y est par contre absent. La vue est fixée sur un DisplayMode égal à Month.

Le DatePicker est un contrôle de saisie d'informations, il demande à l'utilisateur de choisir une date.

L'événement SelectedDateChanged est déclenché lorsque l'utilisateur a fini sa sélection. C'est dans l'attribut SelectedDate que le développeur retrouvera le DateTime choisi :

⚙ Exemple de DatePicker (Xaml)

```

<UserControl x:Class="TestWindowsControls.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

```

```

xmlns:controls="clr-namespace:System.Windows.Controls;
➤ assembly=System.Windows.Controls"
xmlns:data="clr-namespace:System.Windows.Controls;
➤ assembly=System.Windows.Controls.Data"
Width="Auto" Height="Auto">
<Grid x:Name="LayoutRoot" Background="White">
    <controls:DatePicker Name="calendar"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        SelectedDateChanged="DatePicker_SelectedDateChanged"/>
</Grid>
</UserControl>

```



► Fig. 5.20 :
Exemple de DatePicker



Exemple de DatePicker (C#)

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Filmotheque;

namespace TestWindowsControls
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();

```

```

        DateTime Lundi29Juillet85 = new DateTime(1985, 7, 29);
        calendar.DisplayDate = Lundi29Juillet85;
        calendar.DisplayDateStart = Lundi29Juillet85.AddDays(-4);
        calendar.DisplayDateEnd = Lundi29Juillet85.AddDays(4);
    }

    private void DatePicker_SelectedDateChanged(object sender,
        SelectionChangedEventArgs e)
    {
        if((sender as DatePicker).SelectedDate.HasValue)
            DateTime choix =
                (sender as DatePicker).SelectedDate.Value;
    }
}

```

GridSplitter

Un *GridSplitter* est un contrôle utilisateur permettant de redimensionner les tailles des colonnes et des lignes d'une grille :



Exemple de GridSplitter

```

<UserControl x:Class="TestWindowsControls.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:controls="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls"
    xmlns:data="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls.Data"
    Width="250" Height="250">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>

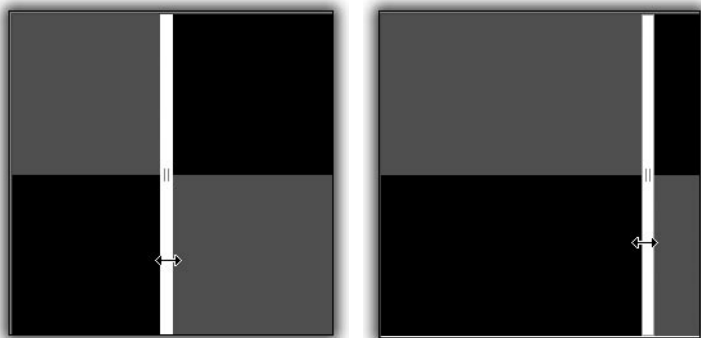
        <Rectangle Grid.Row="0" Grid.Column="0" Fill="Red"/>
        <Rectangle Grid.Row="0" Grid.Column="1" Fill="Black"/>
        <Rectangle Grid.Row="1" Grid.Column="0" Fill="Black"/>
        <Rectangle Grid.Row="1" Grid.Column="1" Fill="Red"/>
    </Grid>
</UserControl>

```

```

<controls:GridSplitter Grid.Column="0" Grid.RowSpan="2"/>
</Grid>
</UserControl>

```



► Fig. 5.21 : Un GridSplitter en action

TabControl et TabItem

Un *TabControl* est un élément de la famille des *Layout*, il permet donc de structurer l'interface.

Ses enfants doivent obligatoirement être des *TabItem*. Chaque *TabItem* est alors classé dans le *TabControl* comme on classerait des feuilles dans un range document.

L'attribut *Header* des *TabItem* permet de préciser le titre. Seul un *TabItem* peut être sélectionné (donc affiché) simultanément.

Lors d'un clic sur le header d'un *TabItem*, ce *TabItem* va directement être sélectionné.

Les événements *SelectionChanged* du *TabControl* et *IsSelectedChanged* des *TabItems* permettent toutefois d'agir sur cette fonctionnalité dans le code applicatif :



Exemple de TabControl

```

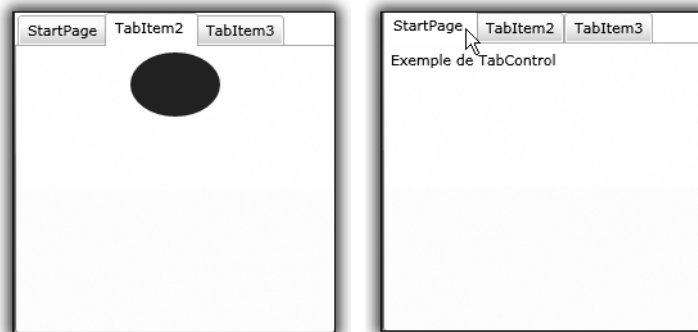
<UserControl x:Class="TestWindowsControls.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:controls="clr-namespace:System.Windows.Controls;
  ➤ assembly=System.Windows.Controls"
  xmlns:data="clr-namespace:System.Windows.Controls;
  ➤ assembly=System.Windows.Controls.Data"
  Width="250" Height="250">
  <Grid x:Name="LayoutRoot" Background="White">
    <controls:TabControl Name="MainTabControl">

```

```

<controls:TabItem Header="StartPage">
  <StackPanel>
    <TextBlock Text="Exemple de TabControl"/>
  </StackPanel>
</controls:TabItem>
<controls:TabItem Header="TabItem2" IsSelected="True">
  <StackPanel>
    <Ellipse Fill="Blue" Height="50" Width="70"/>
  </StackPanel>
</controls:TabItem>
<controls:TabItem Header="TabItem3">
  <StackPanel>
    <TextBlock Text="Exemple de TabControl"/>
  </StackPanel>
</controls:TabItem>
</controls:TabControl>
</Grid>
</UserControl>

```



► Fig. 5.22 : Un TabControl en action

5.5 Le contrôle DataGrid

Le *DataGrid* est le sixième contrôle utilisateur supplémentaire par rapport aux contrôles de base. Il se trouve dans la librairie *System.Windows.Controls.Data*.

Dans l'exemple qui suit, nous allons simplement lier un *DataGrid* à notre collection de films.

Ce *DataGrid* aura la propriété *AutoGeneratedColumn* à *True*. Cette propriété stipule à la plateforme qu'elle doit créer une colonne par propriété des Films :



Exemple de DataGrid auto généré

```

<UserControl x:Class="TestWindowsControls.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

```

```

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:controls="clr-namespace:System.Windows.Controls;
➤ assembly=System.Windows.Controls"
xmlns:data="clr-namespace:System.Windows.Controls;
➤ assembly=System.Windows.Controls.Data"
Width="600" Height="200">
<Grid x:Name="LayoutRoot" Background="White">
    <data:DataGrid Name="dataGrid"
        HeadersVisibility="All"
        ColumnWidth="150"
        RowHeight="20"
        AutoGenerateColumns="True"
        RowBackground="Beige"
        IsReadOnly="False"
        AlternatingRowBackground="LemonChiffon">

    </data:DataGrid>
</Grid>
</UserControl>

```

Les quelques attributs utilisés ici sont :

- **HeadersVisibility** pouvant prendre les valeurs :
 - All. Tous les headers sont visible.
 - Column. Seuls les titres de colonnes sont visibles.
 - None. Aucun des headers n'est visible.
 - Row. Seul le header de sélection de ligne est visible.
- **RowBackground** et **AlternatingRowBackground** permettent de changer la couleur de fond des lignes entre chaque item
- **IsReadOnly** spécifie si les éléments de chaque cellule auto générée sont des éléments d'affichage ou de saisie d'informations :



Exemple de DataGrid (C#)

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;

```

```

using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Filmotheque;

namespace TestWindowsControls
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
            dataGrid.ItemsSource = CollectionDeFilms.Films;
        }
    }
}

```



Titre	Réalisateur	NombreDEtoiles	
Retour vers le Présent 7	Takis Ergopoulos	5	
Il faut sauver le magicien	Orhan Benekhol	1	
La guerre des Chtis	Gautier Legros	4	
2009 l'odisée des Interfax	Ilkay Nemetzakias	3	
SilverLight 2 Ze Movie	Salvador Bargelot	4	

► Fig. 5.23 : DataGrid auto générée

DataGrid non auto généré

L'auto génération a ses limites. En effet, dès que nous aurons des données plus complexes que des chaînes de caractères, des nombres ou des valeurs booléennes, la méthode ToString de ces données sera affichée, nous donnant le même problème qu'un DataBinding direct sans DataTemplates.

Pour créer manuellement des colonnes dans une DataGrid, la librairie *Microsoft.Windows.Controls.Data* offre 3 *UserTemplates* :

- *DataGridTextColumn*. Il s'agit du type de colonnes les plus communes. Elles utilisent un *TextBlock* pour afficher des données et une *TextBox* pour les éditer.
- *DataGridCheckBoxColumn*. Ce type de colonne affiche une *CheckBox*. Comme son nom l'indique, l'attribut *IsEditable* de la *DataGrid* mère de cette colonne est lié à l'attribut *IsEditable* de la *CheckBox*.
- *DataGridTemplateColumn*. Il s'agit du type de colonne par excellence. Il reprend le principe de *DataTemplate* et il est donc totalement versatile.

Utiliser une collection de colonnes

Avant toutes grandes modifications, essayons de recréer sans `AutoGeneratedColumn` le même résultat qu'avec `AutoGeneratedColumn` :



DataGrid avec collection de colonnes

```
<UserControl x:Class="TestWindowsControls.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:controls="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls"
    xmlns:data="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls.Data"
    Width="600" Height="200">
    <Grid x:Name="LayoutRoot" Background="White">
        <data:DataGrid Name="dataGrid"
            HeadersVisibility="All"
            ColumnWidth="150"
            RowHeight="20"
            AutoGenerateColumns="False"
            RowBackground="Beige"
            IsReadOnly="False"
            AlternatingRowBackground="LemonChiffon">
            <data:DataGrid.Columns>
                <data:DataGridTextColumn Header="Titre"
                    Binding="{Binding Path=Titre}" />
                <data:DataGridTextColumn Header="Réalisateur"
                    Binding="{Binding Path=Réalisateur}" />
                <data:DataGridTextColumn Header="Nombre d'étoiles"
                    Binding="{Binding Path=NombreDEtoiles}" />
            </data:DataGrid.Columns>
        </data:DataGrid>
    </Grid>
</UserControl>
```

Titre	Réalisateur	Nombre d'étoiles	
▶ Retour vers le Présent 7	Takis Ergopoulos	5	
Il faut sauver le magicien	Orhan Benekhol	1	
La guerre des Chtis	Gautier Legros	4	
2009 l'odisée des Interfac	Ilkay Nemetzakis	3	
SilverLight 2 Ze Movie	Salvador Bargelot	4	

► Fig. 5.24 : DataGrid avec collection de colonnes

Remarquez d'ors et déjà la subtile différence dans le titre de la colonne représentant le nombre d'étoiles. Ayant la main sur le contenu de son Header, il nous a été possible de le faire passer de *NombreDEtoile* à *Nombre d'étoiles*.

DataGridCheckBoxColumn et DataGridTemplateColumn

Pour aller plus loin dans les exemples, nous devons changer de collection de données. En effet, une collection de données contient uniquement des chaînes de caractères et un nombre ne suffit pas à exploiter la puissance des DataGridTemplateColumn.

Le cas d'étude parfait est une liste de projets.

Un projet est défini par son nom, sa date de départ, sa date butoir, sa liste d'employés.

Un employé est défini par son nom, son prénom, sa date de naissance, sa photo d'identité, son adresse de blog et son email.

Voici le code des sources de données :

Projet.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Collections.Generic;

namespace TestWindowsControls
{
    public class Projet
    {
        private string nom;

        public string Nom
        {
            get { return nom; }
            set { nom = value; }
        }

        private DateTime depart;
```

```

    public DateTime Départ
    {
        get { return départ; }
        set { départ = value; }
    }

    private DateTime buttoir;

    public DateTime Buttoir
    {
        get { return buttoir; }
        set { buttoir = value; }
    }

    private List<Employé> équipe;

    public List<Employé> Equipe
    {
        get { return équipe; }
        set { équipe = value; }
    }

    public Projet() { }
}

```



Employé.cs

```

using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace TestWindowsControls
{
    public class Employé
    {
        private string nom;

        public string Nom

```

```

    {
        get { return nom; }
        set { nom = value; }
    }

    private string prenom;

    public string Prenom
    {
        get { return prenom; }
        set { prenom = value; }
    }

    private Uri photo;

    public Uri Photo
    {
        get { return photo; }
        set { photo = value; }
    }

    private Uri blog;

    public Uri Blog
    {
        get { return blog; }
        set { blog = value; }
    }

    private string email;

    public string Email
    {
        get { return email; }
        set { email = value; }
    }

    public Employé() { }
}

```



CollectionDeProjet.cs

```

using System;
using System.Net;
using System.Windows;

```

```

using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Collections.Generic;

namespace TestWindowsControls
{
    public static class CollectionDeProjet
    {
        public static List<Projet> Projets = new List<Projet>()
        {
            new Projet()
            {
                Nom="OpenApp",
                Départ=new DateTime(2009,1,12),
                Buttoir=new DateTime(2009,2,15),
                Equipe = new List<Employé>()
                {
                    CollectionDeProjet.Simon,
                    CollectionDeProjet.Caroline
                }
            },

            new Projet()
            {
                Nom="Livre SL2",
                Départ=new DateTime(2006,8,5),
                Buttoir=new DateTime(2000,1,10),
                Equipe = new List<Employé>()
                {
                    CollectionDeProjet.Loïc,
                    CollectionDeProjet.Simon
                }
            },

            new Projet()
            {
                Nom="Jubbeo",
                Départ=new DateTime(2009,3,6),
                Buttoir=new DateTime(2009,8,21),
                Equipe = new List<Employé>()
                {
                    CollectionDeProjet.Florent,

```

```

        CollectionDeProjet.Simon,
        CollectionDeProjet.Loïc
    }
},

new Projet()
{
    Nom="Wipus",
    Départ=new DateTime(2006,7,1),
    Buttoir=new DateTime(2009,4,8),
    Equipe = new List<Employé>()
    {
        CollectionDeProjet.Simon,
        CollectionDeProjet.Loïc,
        CollectionDeProjet.X
    }
}
};

public static Employé Loïc = new Employé()
{
    Nom = "Loïc",
    Prenom = "Bar",
    Email = "loic.bar@wipus.com",
    Photo = new Uri("Employe/Loic.jpg", UriKind.Relative),
    Blog = new Uri("http://www.loicbar.com",UriKind.Absolute)
};

public static Employé Simon = new Employé()
{
    Nom = "Simon",
    Prenom = "Boigelot",
    Email = "simon.boigelot@wipus.com",
    Photo = new Uri("Employe/Simon.jpg", UriKind.Relative),
    Blog = new Uri("http://www.simonboigelot.com", UriKind.Absolute)
};

public static Employé Florent = new Employé()
{
    Nom = "Florent",
    Prenom = "G.",
    Email = "MisterFlo@provider.com",
    Photo = new Uri("Employe/Buddy.JPG", UriKind.Relative),
    Blog = new Uri("http://fake.MisterFlo.com", UriKind.Absolute)
};

```

```

public static Employé Caroline = new Employé()
{
    Nom = "Caroline",
    Prenom = "L.",
    Email = "caroline@wipus.com",
    Photo = new Uri("Employe/Buddy.JPG", UriKind.Relative),
    Blog = new Uri("http://fake.KRO.com", UriKind.Absolute)
};

public static Employé X = new Employé()
{
    Nom = "",
    Prenom = "X",
    Email = "x@wipus.com",
    Photo = new Uri("Employe/Buddy.JPG", UriKind.Relative),
    Blog = new Uri("http://fake.X.com", UriKind.Absolute)
};
}

```

En guise d'illustration du problème dû à l'auto génération, voici ce que donnerait une *DataGrid* auto générée dont l'*itemsSource* est *CollectionDeProjet.Projets*.

Nom	Départ	Buttoir	Equipe
▶ OpenApp	1/12/2009 12:00:00 AM	2/15/2009 12:00:00 AM	System.Collections.Gener
Livre SL2	8/5/2006 12:00:00 AM	1/10/2000 12:00:00 AM	System.Collections.Gener
Jubbeo	3/6/2009 12:00:00 AM	8/21/2009 12:00:00 AM	System.Collections.Gener
Wipus	7/1/2006 12:00:00 AM	4/8/2009 12:00:00 AM	System.Collections.Gener

► Fig. 5.25 : Problème du à *AutoGeneratedColumn*

Les dates représentées dans cette *DataGrid* ont un aspect à la limite de l'illisible. Quant à la liste d'employés, impossible de savoir ce qu'elle contient.

Reproduisons d'abord ce résultat :



Reproduction du résultat auto généré

```

<UserControl x:Class="TestWindowsControls.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:controls="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls"
    xmlns:data="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls.Data"

```

```

Width="Auto" Height="Auto">
<Grid x:Name="LayoutRoot" Background="White">
  <data:DataGrid Name="dataGrid"
    HeadersVisibility="All"
    ColumnWidth="150"
    RowHeight="20"
    AutoGenerateColumns="False"
    RowBackground="Beige"
    IsReadOnly="False"
    AlternatingRowBackground="LemonChiffon">
    <data:DataGrid.Columns>
      <data:DataGridTextColumn Header="Nom du Projet"
        Binding="{Binding Path=Nom}" />
      <data:DataGridTextColumn Header="Date de début"
        Binding="{Binding Path=Départ}" />
      <data:DataGridTextColumn Header="Date buttoir"
        Binding="{Binding Path=Buttoir}" />
      <data:DataGridTextColumn Header="Equipe"
        Binding="{Binding Path=Equipe}" />
    </data:DataGrid.Columns>
  </data:DataGrid>
</Grid>
</UserControl>

```

Nom du Projet	Date de début	Date buttoir	Equipe
OpenApp	1/12/2009 12:00:00 AM	2/15/2009 12:00:00 AM	System.Collections.Gener
Livre SL2	8/5/2006 12:00:00 AM	1/10/2000 12:00:00 AM	System.Collections.Gener
Jubbeo	3/6/2009 12:00:00 AM	8/21/2009 12:00:00 AM	System.Collections.Gener
Wipus	7/1/2006 12:00:00 AM	4/8/2009 12:00:00 AM	System.Collections.Gener

► Fig. 5.26 : Reproduction du résultat auto généré

Modification de l'affichage des dates

Pour afficher des dates, rien de mieux qu'un *DatePicker*.

Nous allons remplacer dans le code XAML de l'application les deux *DataGridTextColumns* **Date de début** et **Date buttoir** par des *DataGridTemplateColumns* contenant un *DatePicker*.



Exemple de DataGridTemplateColumn DatePicker

```

<data:DataGridTemplateColumn Header="Date de début">
  <data:DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
      <controls:DatePicker

```

```

        SelectedDate="{Binding Départ, Mode=OneWay}" />
    </DataTemplate>
</data:DataGridTemplateColumn.CellTemplate>
<data:DataGridTemplateColumn.CellEditingTemplate>
    <DataTemplate>
        <controls:DatePicker
            SelectedDate="{Binding Départ, Mode=TwoWay}" />
    </DataTemplate>
</data:DataGridTemplateColumn.CellEditingTemplate>
</data:DataGridTemplateColumn>

```

Deux attributs de Template sont à redéfinir dans cet *UserControl*, le *CellTemplate* (Template utilisé en mode d’affichage de données) et le *CellEditingTemplate* (Template utilisé en mode d’édition des données)

Le remplacement de ces *DataTemplate* à la sauce *DataGrid* contribue déjà beaucoup à la propreté de notre interface :



► Fig. 5.27 : DataGrid avec DataGridTemplateColumn DatePicker

Modification de l’affichage de la liste d’employés

Pour ce qui est de la liste d’employés, une bonne façon de s’entraîner à la manipulation des *DataGrid* est d’écrire *DataGrid* en tant que redéfinition d’une cellule par une autre partie de *DataGrid*.

Débutons avec l’attribut *AutoGeneratedColumn* à *True*. Il est évident qu’il faudra changer cela le plus vite possible :



DataGridTemplateColumn contenant une DataGrid auto générée

```

<data:DataGridTemplateColumn Header="Equipe">
    <data:DataGridTemplateColumn.CellTemplate>
        <DataTemplate>

```



```
<data:DataGrid AutoGenerateColumns="True"
ItemsSource="{Binding Path=Equipe}"
IsReadOnly="True">

    <!--<data:DataGrid.Columns>
<data:DataGridTextColumn
    Header="Nom"
    Binding="{Binding Nom}"/>
</data:DataGrid.Columns>-->

    </data:DataGrid>
</DataTemplate>
</data:DataGridTemplateColumn.CellTemplate>
<data:DataGridTemplateColumn.CellEditingTemplate>
    <DataTemplate>
        <data:DataGrid AutoGenerateColumns="True"
            ItemsSource="{Binding Path=Equipe}"
            IsReadOnly="False">

            </data:DataGrid>
        </DataTemplate>
    </data:DataGridTemplateColumn.CellEditingTemplate>
</data:DataGridTemplateColumn>
```

Le résultat est déjà très satisfaisant :

Nom du Projet	Date de début	Date butoir	Equipe				
			Nom	Prenom	Photo	Blog	Email
OpenApp	12/01/2009	15/02/2009	Simon	Boigelot	Employe/Simon.jpg	http://www.simonboigelot.com	simon.boigelot@wipus.com
			Caroline	L.	Employe/Buddy.JPG	http://fake.KRO.com	caroline@wipus.com
Livre SL2	5/08/2006	10/01/2000	Loic	Bar	Employe/Loic.jpg	http://www.loicbar.com	loic.bar@wipus.com
			Simon	Boigelot	Employe/Simon.jpg	http://www.simonboigelot.com	simon.boigelot@wipus.com
Jubbeo	6/03/2009	21/08/2009	Florent	G.	Employe/Buddy.JPG	http://fake.MisterFlo.com	MisterFlo@provider.com
			Simon	Boigelot	Employe/Simon.jpg	http://www.simonboigelot.com	simon.boigelot@wipus.com
Wipus	1/07/2006	8/04/2009	Loic	Bar	Employe/Loic.jpg	http://www.loicbar.com	loic.bar@wipus.com
			Simon	Boigelot	Employe/Simon.jpg	http://www.simonboigelot.com	simon.boigelot@wipus.com
			Loic	Bar	Employe/Loic.jpg	http://www.loicbar.com	loic.bar@wipus.com
			X		Employe/Buddy.JPG	http://fake.X.com	x@wipus.com

► Fig. 5.28 : DataGridTemplateColumn contenant une DataGrid auto générée

Pour rendre le tout encore plus beau, nous allons créer pour chaque employé une mini carte de visite.

C'est maintenant que l'on passe à `False` la valeur `AutoGenerateColumn` de notre *DataGrid* Equipe, elle-même *DataTemplate* d'une *DataGridTemplateColumn* de notre *DataGrid* principale :



Carte de visite pour Employés

```
<data:DataGrid AutoGenerateColumns="False"
    ItemsSource="{Binding Path=Equipe}"
    IsReadOnly="True">

    <data:DataGrid.Columns>
        <data:DataGridTemplateColumn>
            <data:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <Image Source="{Binding Path=ImageSourcePhoto}"
                            Height="50" Width="50"
                            Stretch="UniformToFill"/>
                        <StackPanel Orientation="Vertical">
                            <StackPanel Orientation="Horizontal">
                                <TextBlock Text="{Binding Path=Prenom}"
                                    Margin="5"/>
                                <TextBlock Text="{Binding Path=Nom}"
                                    Margin="5"/>
                            </StackPanel>
                            <HyperlinkButton Content="{Binding Blog}"/>
                            <HyperlinkButton Content="{Binding Email}"/>
                        </StackPanel>
                    </StackPanel>
                </DataTemplate>
            </data:DataGridTemplateColumn.CellTemplate>
        </data:DataGridTemplateColumn>
    </data:DataGrid.Columns>

</data:DataGrid>
```

Nom du Projet	Date de début	Date buttoir	Equipe
OpenApp	12/01/2009	15/02/2009	 Boigelot Simon http://www.simonboigelot.com simon.boigelot@wipus.com
			 L. Caroline http://fake.KRO.com caroline@wipus.com
Livre SL2	5/08/2006	10/01/2000	 Bar Loic http://www.loicbar.com loic.bar@wipus.com

► Fig. 5.29 :
DataGrid complètement
customisée

Il a été nécessaire d'ajouter une propriété à la définition de classe *Etudiant*.

Cette propriété est *ImageSourcePhoto* de type *ImageSource*. En effet, l'attribut *Source* d'une *Image* est une *ImageSource* et non un *URI* :

Propriété *ImageSourcePhoto*

```
namespace TestWindowsControls
{
    public class Employé
    {
        (...)
        public ImageSource ImageSourcePhoto
        {
            get
            {
                return new BitmapImage(Photo);
            }
        }
        (...)
    }
}
```

Listing de l'interface complète de la *DataGrid* templatisée

DataGrid Templatisée

```
<UserControl x:Class="TestWindowsControls.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:controls="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls"
    xmlns:data="clr-namespace:System.Windows.Controls;
    ➤ assembly=System.Windows.Controls.Data"
    Width="Auto" Height="Auto">
    <Grid x:Name="LayoutRoot" Background="White">
        <data:DataGrid Name="dataGrid"
            HeadersVisibility="All"
            AutoGenerateColumns="False"
            RowBackground="Beige"
            IsReadOnly="False"
            AlternatingRowBackground="LemonChiffon">
            <data:DataGrid.Columns>
                <data:DataGridTextColumn Header="Nom du Projet"
```

```

Binding="{Binding Path=Nom}" />

<data:DataGridTemplateColumn Header="Date de début">
  <data:DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
      <controls:DatePicker VerticalAlignment="Top"
        SelectedDate="{Binding Départ, Mode=OneWay}" />
    </DataTemplate>
  </data:DataGridTemplateColumn.CellTemplate>
  <data:DataGridTemplateColumn.CellEditingTemplate>
    <DataTemplate>
      <controls:DatePicker VerticalAlignment="Top"
        SelectedDate="{Binding Départ, Mode=TwoWay}" />
    </DataTemplate>
  </data:DataGridTemplateColumn.CellEditingTemplate>
</data:DataGridTemplateColumn>

<data:DataGridTemplateColumn Header="Date buttoir">
  <data:DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
      <controls:DatePicker VerticalAlignment="Top"
        SelectedDate="{Binding Buttoir, Mode=OneWay}" />
    </DataTemplate>
  </data:DataGridTemplateColumn.CellTemplate>
  <data:DataGridTemplateColumn.CellEditingTemplate>
    <DataTemplate>
      <controls:DatePicker VerticalAlignment="Top"
        SelectedDate="{Binding Buttoir, Mode=TwoWay}" />
    </DataTemplate>
  </data:DataGridTemplateColumn.CellEditingTemplate>
</data:DataGridTemplateColumn>

<data:DataGridTemplateColumn Header="Equipe">
  <data:DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
<data:DataGrid AutoGenerateColumns="False"
  ItemsSource="{Binding Path=Equipe}"
  IsReadOnly="True">

<data:DataGrid.Columns>
  <data:DataGridTemplateColumn>
    <data:DataGridTemplateColumn.CellTemplate>
      <DataTemplate>
        <StackPanel Orientation="Horizontal">
          <Image Source="{Binding Path=ImageSourcePhoto}"
            Height="50" Width="50"

```

```

        Stretch="UniformToFill"/>
        <StackPanel Orientation="Vertical">
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding Path=Prenom}"
                Margin="5"/>
                <TextBlock Text="{Binding Path=Nom}"
                Margin="5"/>
            </StackPanel>
            <HyperlinkButton Content="{Binding Blog}"/>
            <HyperlinkButton Content="{Binding Email}"/>
        </StackPanel>
    </StackPanel>
</DataTemplate>
</data:DataGridTemplateColumn.CellTemplate>
</data:DataGridTemplateColumn>
</data:DataGrid.Columns>
</data:DataGrid>
</DataTemplate>
</data:DataGridTemplateColumn.CellTemplate>
</data:DataGridTemplateColumn>
</data:DataGrid.Columns>
</data:DataGrid>
</Grid>
</UserControl>

```

5.6 Les contrôles Silverlight Toolkit de CodePlex

Les derniers contrôles que ce livre aborde sont les contrôles fournis par le *Silverlight ToolKit de CodePlex*.

Ce *ToolKit* ajoute 12 nouveaux *UserControl* aux contrôles utilisateur présents dans la version de base de Silverlight.

Ces 12 contrôles utilisateur se répartissent en trois catégories :

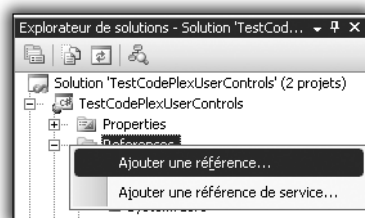
- les contrôles utilisateur de saisie d'informations ;
- les contrôles utilisateur de structuration d'informations ;
- les contrôles utilisateur de styles et thèmes.

Pour utiliser ces nouveaux contrôles, il faut préalablement télécharger la librairie sur le site de CodePlex : <http://www.codeplex.com/Silverlight/>.

Ensuite, pour ajouter à votre projet Silverlight les différentes DLL en tant que références :

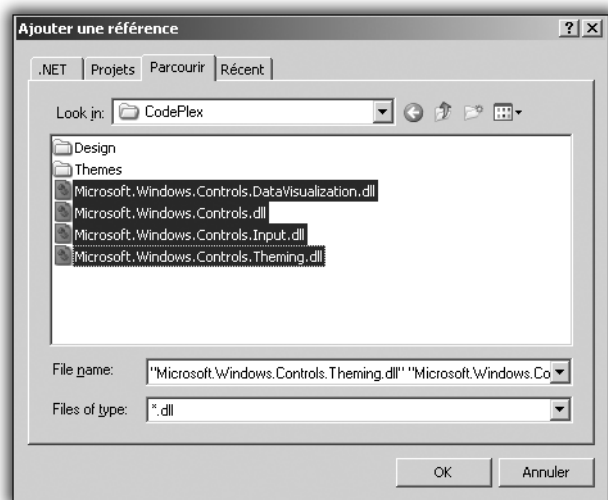
- 1 Cliquez du bouton droit sur **Reference** dans l'Explorateur de solution.

- 2 Sélectionnez l'action **Ajouter une référence**.



► Fig. 5.30 :
Add Reference

- 3 Dans la boîte de dialogue **Ajouter une référence**, sélectionnez l'onglet **Parcourir**.
- 4 Naviguez jusqu'à l'emplacement des DLL et sélectionnez les fichiers *Microsoft.Windows.Controls.Theming.dll*, *Microsoft.Windows.Controls.DataVisualization.dll*, *Microsoft.Windows.Controls.dll* et *Microsoft.Windows.Controls.Input.dll*.



► Fig. 5.31 :
Boîte de dialogue
Ajouter une référence

- 5 Cliquez sur OK.

Vous trouverez tous les exemples et toutes les informations nécessaires à l'utilisation de ces nouveaux *UserControls* dans le fichier ZIP fourni par *CodePlex*.

5.7 Check-list

Dans ce chapitre, nous avons amélioré considérablement nos compétences en XAML, étudié des moyens efficaces pour rendre un code d'interface plus lisible ainsi que d'autres moyens, tout aussi efficaces, pour obtenir l'effet inverse.

Le juste milieu entre un code propre et une expérience utilisateur riche ne semble pas encore accessible ; seule l'expérience pourra vous aider sur cette voie.

Sachez que rien n'est impossible, en Silverlight, ou presque. Tout ce que vous pouvez imaginer, vous pouvez le coder ; la plateforme .Net vous y aidera.

6



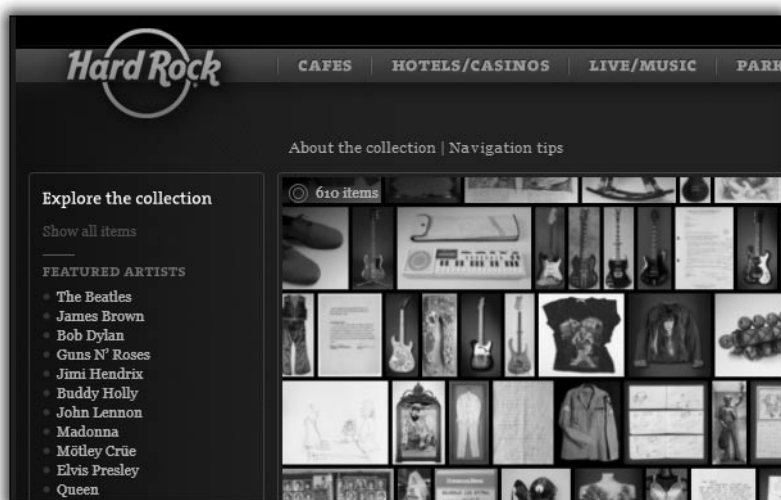
6.1 Introduction à Deepzoom	274
6.2 Fonctionnement de Deepzoom	275
6.3 Deepzoom par l'exemple	279
6.4 Deepzoom et Virtual Earth	288
6.5 Check-list	289

Découvrir Deepzoom

Deepzoom est une technologie introduite dans Silverlight 2. C'est un des points forts de Silverlight. En effet, *Deepzoom* permet une gestion optimale des images. Nous allons découvrir dans ce chapitre comment utiliser cette technologie hors du commun au travers d'un exemple simple mais efficace avec *Deepzoom*.

6.1 Introduction à Deepzoom

DeepZoom permet d'effectuer un zoom performant sur des images presque arbitrairement de grandes tailles dans *Silverlight*. Les images peuvent être affichées à une échelle très petite et très grande sans affecter les performances de l'application qui affiche l'image. La seule propriété qui affecte les performances est le nombre de pixels à afficher à l'écran. Mais il existe des cas déjà largement diffusés sur Internet qui utilisent des milliards de pixels. C'est le cas du hard rock que vous pouvez trouver à cette adresse : <http://memorabilia.hardrock.com/>.



► Fig. 6.1 : Hard rock

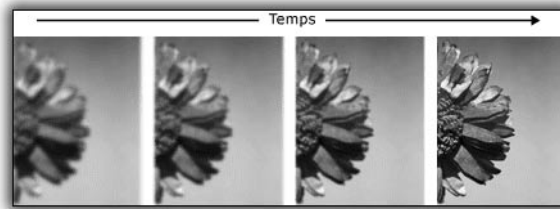
Youtube, *Dailymotion* et d'autres se sont mis à la haute résolution. *Deepzoom* est là pour que vous puissiez faire de même avec les images. Souvent, les photos sur le Web sont de mauvaise qualité ; on peut très rarement zoomer correctement sur une image. Vous pouvez utiliser *deepzoom* pour afficher des images de très grande qualité. Vous pouvez ainsi imaginer un site de visualisation d'images haute résolution pour les photos de vacances, de mariage ou les photos professionnelles.

Deepzoom permet également une visualisation panoramique d'un paysage ou d'une maison. Imaginez cela pour des sites consacrés à l'immobilier !

Cela permet également de nouveaux modèles de publicité. En effet, le zoom peut apporter des informations supplémentaires sur un produit mis en publicité.

6.2 Fonctionnement de Deepzoom

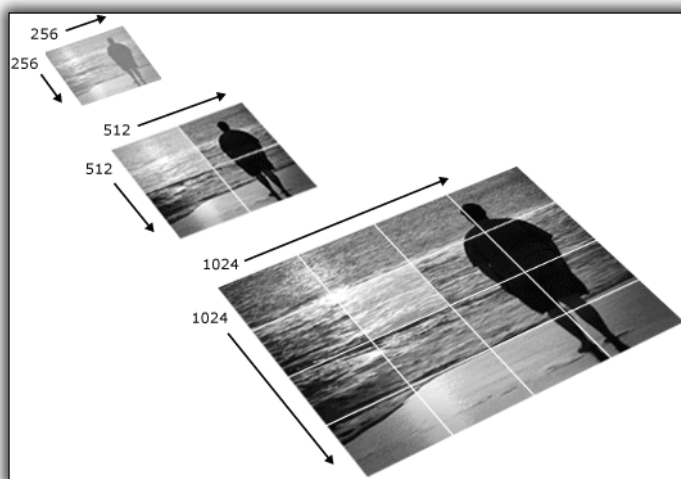
Deepzoom charge en fait les images en basse résolution et charge les résolutions plus grandes au fur et à mesure. C'est pour cette raison que le chargement de l'image est très rapide. Quand on regarde les sondages, ce qui embête le plus un utilisateur est de devoir attendre pour le chargement d'une page. Ici on charge les informations de base de l'image ; au début, l'image reste floue mais bien présente. Au fur et à mesure, l'image s'éclaircit et avant même que l'utilisateur n'ait eu le temps de commencer à zoomer, l'image est chargée entièrement. Ceci permet d'avoir une expérience utilisateur unique et vraiment satisfaisante.



► Fig. 6.2 :
Image dans le temps

L'image haute résolution est en fait découpée en plusieurs images. Microsoft appelle cela la pyramide d'images. Une pyramide d'images décompose une image en fragments de 256 x 256 d'images JPG ou PNG (dans ce cas, la taille est arbitraire et peut être modifiée) et stocke également des versions de résolution inférieure de l'image dans les fragments. Chaque fragment est stocké à l'intérieur d'un fichier distinct et chaque niveau de la pyramide est stocké à l'intérieur de dossiers distincts. L'image ci-après décrit schématiquement le fonctionnement de la pyramide d'images. L'image elle-même est disponible en pleine résolution dans le bas de la pyramide (voir l'image ci-après) et les versions de résolution inférieure jusqu'à 4 x 4 pixels sont stockées avec l'image pleine résolution. Les images à chaque niveau de la pyramide sont stockées dans des fragments de 256 x 256 pixels (lignes blanches dans les images). *DeepZoom* peut ainsi extraire uniquement les fragments requis pour la taille actuelle de l'image à l'écran, au lieu de télécharger toute l'image. Par exemple, si vous effectuez un zoom avant pour afficher uniquement la partie centrale en surbrillance de l'image, *DeepZoom* charge uniquement les fragments en surbrillance, plutôt que toute l'image au format 1 024 x 1 024.

La création manuelle de ces pyramides peut se révéler fastidieuse. Par conséquent, il est recommandé d'utiliser un outil permettant de convertir les images en une pyramide d'images. Par exemple, pour ce faire, vous pouvez utiliser *DeepZoom Composer*. Vous trouverez cet utilitaire sur le site de Silverlight.net.



► Fig. 6.3 :
Pyramide d'images

Le format de fichier qui sert à accéder à la pyramide d'images utilise un schéma *XML*. À nouveau, vous pouvez générer ce format de fichier à l'aide de *DeepZoom Composer*. Toutefois, si vous souhaitez exercer un contrôle plus précis sur le format de fichier, vous pouvez créer manuellement le code *XML* ou apporter des modifications manuelles à un fichier généré par un outil.

Vous pouvez vous-même créer un les fichiers *XML*. Mais cela est loin d'être simple.

D'abord, il génère un fichier de *metadata* :

```
<?xml version="1.0"?>
<Metadata version="1">
  <AspectRatio>0.733841659419689</AspectRatio>
  <Image>
    <FileName>C:\Documents and Settings\samlan\Desktop\Labs\
    ➤ DeepZoomCollections\DeepZoomOutput\DeepZoomComposer\source
    ➤ images\tree_blossoms.jpg</FileName>
    <x>0</x>
    <y>0</y>
    <Width>0.429715926819635</Width>
    <Height>0.4739336492891</Height>
    <ZOrder>1</ZOrder>
    <Tag />
  </Image>
  <Image>
    <FileName>C:\Documents and
    ➤ Settings\samlan\Desktop\Labs\DeepZoomCollections\
    ➤ DeepZoomOutput\DeepZoomComposer\source
    ➤ images\guy_by_the_beach.jpg</FileName>
```

```

    <x>0.55991053434817</x>
    <y>0</y>
    <Width>0.44008946565183</Width>
    <Height>0.4739336492891</Height>
    <ZOrder>2</ZOrder>
    <Tag />
  </Image>
  <Image>
    <FileName>C:\Documents and Settings\samlan\Desktop\Labs\
    ➤ DeepZoomCollections\DeepZoomOutput\DeepZoomComposer\source
    ➤ images\licorice.jpg</FileName>
    <x>0</x>
    <y>0.5260663507109</y>
    <Width>0.431362685784476</Width>
    <Height>0.4739336492891</Height>
    <ZOrder>3</ZOrder>
    <Tag />
  </Image>
  <Image>
    <FileName>C:\Documents and Settings\samlan\Desktop\Labs\
    ➤ DeepZoomCollections\DeepZoomOutput\DeepZoomComposer\source
    ➤ images\flower.jpg</FileName>
    <x>0.579552839832946</x>
    <y>0.5260663507109</y>
    <Width>0.420447160167053</Width>
    <Height>0.4739336492891</Height>
    <ZOrder>4</ZOrder>
    <Tag />
  </Image>
</Metadata>

```

Ainsi qu'un deuxième fichier presque identique mais avec un ratio :

```

<?xml version="1.0"?>
<SceneGraph version="1">
  <AspectRatio>0.733841659419689</AspectRatio>
  <SceneNode>
    <FileName>C:\Documents and Settings\samlan\Desktop\Labs\
    ➤ DeepZoomCollections\DeepZoomOutput\DeepZoomComposer\source
    ➤ images\tree_blossoms.jpg</FileName>
    <x>0</x>
    <y>0</y>
    <Width>0.429715926819635</Width>
    <Height>0.4739336492891</Height>
    <ZOrder>1</ZOrder>
  </SceneNode>
  <SceneNode>

```

```

<FileName>C:\Documents and Settings\samlan\Desktop\Labs\
  ➤ DeepZoomCollections\DeepZoomOutput\DeepZoomComposer\source
  ➤ images\guy_by_the_beach.jpg</FileName>
<x>0.55991053434817</x>
<y>0</y>
<Width>0.44008946565183</Width>
<Height>0.4739336492891</Height>
<ZOrder>2</ZOrder>
</SceneNode>
<SceneNode>
  <FileName>C:\Documents and Settings\samlan\Desktop\Labs\
    ➤ DeepZoomCollections\DeepZoomOutput\DeepZoomComposer\source
    ➤ images\licorice.jpg</FileName>
  <x>0</x>
  <y>0.5260663507109</y>
  <Width>0.431362685784476</Width>
  <Height>0.4739336492891</Height>
  <ZOrder>3</ZOrder>
</SceneNode>
<SceneNode>
  <FileName>C:\Documents and Settings\samlan\Desktop\Labs\
    ➤ DeepZoomCollections\DeepZoomOutput\DeepZoomComposer\source
    ➤ images\flower.jpg</FileName>
  <x>0.579552839832946</x>
  <y>0.5260663507109</y>
  <Width>0.420447160167053</Width>
  <Height>0.4739336492891</Height>
  <ZOrder>4</ZOrder>
</SceneNode>
</SceneGraph>

```

Il génère enfin un fichier contenant des informations sur les images et des références vers les *metadata* de chacune des images :

```

<?xml version="1.0" encoding="UTF-8"?>
<Collection MaxLevel="8" TileSize="256" Format="png" NextItemId="4"
  xmlns="http://schemas.microsoft.com/deepzoom/2008">
  <Items>
    <I Id="0" N="0" IsPath="1"
      Source="dzc_output_images/tree_blossoms.xml">
      <Size Width="515" Height="774" />
      <Viewport Width="2.32711864" X="0" Y="0" />
    </I>
    <I Id="1" N="1" IsPath="1"
      Source="dzc_output_images/guy_by_the_beach.xml">
      <Size Width="569" Height="835" />
      <Viewport Width="2.2722652" X="-1.2722652" Y="0" />
    </I>
  </Items>
</Collection>

```

```
<I Id="2" N="2" IsPath="1" Source="dzc_output_images/licorice.xml">
  <Size Width="531" Height="795" />
  <Viewport Width="2.318235" X="0" Y="-1.66186452" />
</I>
<I Id="3" N="3" IsPath="1" Source="dzc_output_images/flower.xml">
  <Size Width="541" Height="831" />
  <Viewport Width="2.37842" X="-1.37842011" Y="-1.70500922" />
</I>
</Items>
</Collection>
```

Ce fichier référence par exemple les informations sur *flower* :

```
<?xml version="1.0" encoding="UTF-8"?>
<Image TileSize="256" Overlap="1" Format="png"
  xmlns="http://schemas.microsoft.com/deepzoom/2008">
  <Size Width="541" Height="831"/>
</Image>
```

Vous voyez également une série de dossiers : 1, 2, 3, 4, etc. Ces dossiers contiennent une ou plusieurs images. Elles représentent l'élément de plus en plus grand.

6.3 Deepzoom par l'exemple

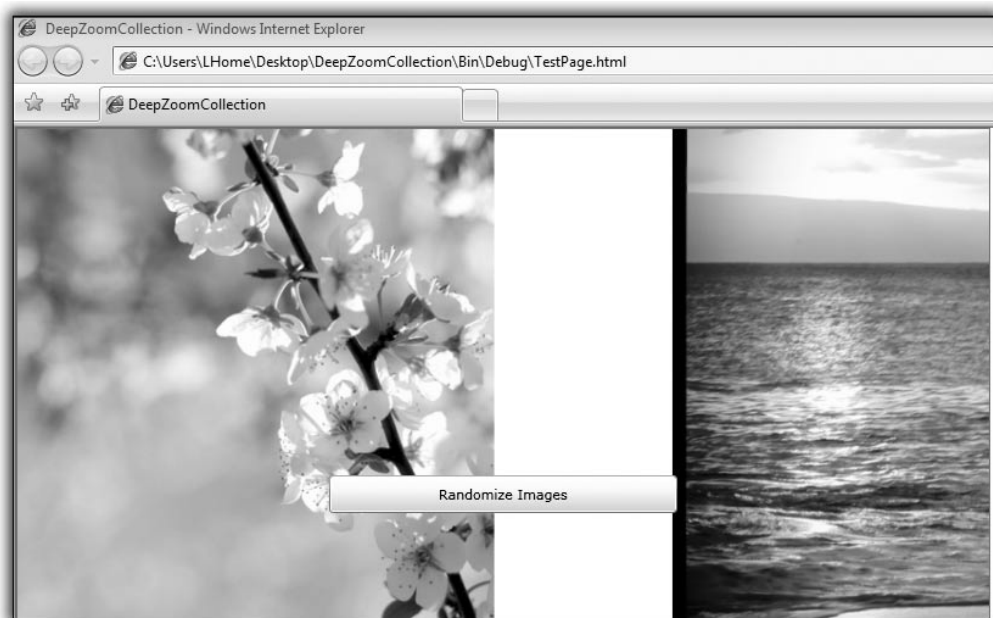
Pour bien comprendre *Deepzoom*, il faut l'utiliser. Voici en image l'application *Silverlight* que nous allons créer :

Pour commencer, il faut définir le *XAML* pour cette application. Si on regarde attentivement, on remarque seulement deux éléments pertinents :

- une zone pour les images ;
- un bouton.

La zone pour les images est un peu particulière ; c'est une zone spécifique à *Deepzoom* que vous n'avez probablement jamais rencontrée auparavant. C'est un objet *XAML* *MultiScaleImage* :

```
<MultiScaleImage Height="600" x:Name="msi"
  Source="GeneratedImages/dzc_output.xml" Width="800"/>
```



► Fig. 6.4 : Application Silverlight de test

Cette zone fait référence au fichier XML que nous avons vu précédemment et qui ressemble à ceci :

```
<?xml version="1.0" encoding="UTF-8"?>
<Collection MaxLevel="8" TileSize="256" Format="png" NextItemId="4"
  xmlns="http://schemas.microsoft.com/deepzoom/2008">
  <Items>
    <I Id="0" N="0" IsPath="1"
      Source="dzc_output_images/tree_blossoms.xml">
      <Size Width="515" Height="774" />
      <Viewport Width="2.32711864" X="0" Y="0" />
    </I>
  </Items>
  ...
```

Une fois que vous avez assimilé le nouvel élément XAML, le code n'est pas très compliqué :

```
<UserControl x:Class="DeepZoomProject.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
```



```

Width="800" Height="600">
<Grid x:Name="LayoutRoot" Background="White">
    <Border BorderBrush="#FF727272" BorderThickness="1,1,1,1">
        <MultiScaleImage Height="600" x:Name="msi"
            Source="GeneratedImages/dzc_output.xml" Width="800"/>
    </Border>
    <Button Height="31" Width="286" Content="Randomize Images"
        Click="Arrange_Click"/>
</Grid>
</UserControl>

```

Sur notre `MultiScaleImage`, nous allons déclarer une série d'événements qui nous permettront de savoir ce que fait l'utilisateur sur l'application (clic, déplacement, etc.). Ceci se fera dans le constructeur de notre page (dans le *code behind*, code attaché) :

```

msi.MouseMove += delegate(object sender, MouseEventArgs e)
{
    if (mouseButtonPressed)
    {
        mouseIsDragging = true;
    }
    this.lastMousePos = e.GetPosition(this.msi);
};

msi.MouseLeftButtonDown += delegate(object sender, MouseButtonEventArgs e)
{
    mouseButtonPressed = true;
    mouseIsDragging = false;
    dragOffset = e.GetPosition(this);
    currentPosition = msi.ViewportOrigin;
};

msi.MouseLeave += delegate(object sender, MouseEventArgs e)
{
    mouseIsDragging = false;
};

msi.MouseLeftButtonUp += delegate(object sender, MouseButtonEventArgs e)
{
    mouseButtonPressed = false;
    if (mouseIsDragging == false)
    {
        bool shiftDown = (Keyboard.Modifiers & ModifierKeys.Shift) ==
            ➡ ModifierKeys.Shift;

        ZoomFactor = 2.0;
        if (shiftDown) ZoomFactor = 0.5;
    }
}

```

```

        Zoom(ZoomFactor, this.lastMousePos);
    }
    mouseIsDragging = false;
};

msi.MouseMove += delegate(object sender, MouseEventArgs e)
{
    if (mouseIsDragging)
    {
        Point newOrigin = new Point();
        newOrigin.X = currentPosition.X - (((e.GetPosition(msi).X -
            dragOffset.X) / msi.ActualWidth) * msi.ViewportWidth);
        newOrigin.Y = currentPosition.Y - (((e.GetPosition(msi).Y -
            dragOffset.Y) / msi.ActualHeight) * msi.ViewportWidth);
        msi.ViewportOrigin = newOrigin;
    }
};

```

On voit apparaître l'utilisation d'une fonction Zoom, fonction que l'on utilise pour atteindre un point :

```

public void Zoom(double zoom, Point pointToZoom)
{
    Point logicalPoint = this.msi.ElementToLogicalPoint(pointToZoom);
    this.msi.ZoomAboutLogicalPoint(zoom, logicalPoint.X, logicalPoint.Y);
}

```

Tout ce code ne prend malheureusement pas en compte la molette de la souris qui est pourtant très utilisée. Pour cela, nous pouvons créer une classe qui prendra en compte cette molette. Cette classe a été définie au départ par Pete Bois et adaptée au projet. Vous trouverez le code de cette classe à la fin du chapitre.

Vous pouvez vous attacher aux événements de la molette de la manière suivante :

```

new MouseWheelHelper(msi).Moved += delegate(object sender,
➡ MouseWheelEventArgs e)
{
    e.Handled = true;
    if (e.Delta > 0)
        ZoomFactor = 1.2;
    else
        ZoomFactor = .80;

    Zoom(ZoomFactor, this.lastMousePos);
};

```

Dans notre programme, il faut juste gérer correctement le clic sur le bouton au milieu de notre application *Silverlight*. Pour cela, une fonction intercepte l'événement `click` de notre bouton :

```
private void Arrange_Click(object sender, RoutedEventArgs e)
{
    ArrangeIntoGrid();
}
```

Cette fonction va réarranger les images. Nous devons créer une animation. Vous avez vu dans un chapitre précédent ce qu'était les *storyboards*. Nous allons ici créer un *storyboard* de façon dynamique, c'est-à-dire directement dans le code :

```
Storyboard moveStoryboard = new Storyboard();
```

Ensuite, il faut créer l'animation et les frames qui vont intervenir dedans :

```
// Create Animation
PointAnimationUsingKeyFrames moveAnimation = new
➤ PointAnimationUsingKeyFrames();

// Create Keyframe
SplinePointKeyFrame startKeyframe = new SplinePointKeyFrame();
startKeyframe.Value = currentPosition;
startKeyframe.KeyTime = KeyTime.FromTimeSpan(TimeSpan.Zero);

startKeyframe = new SplinePointKeyFrame();
startKeyframe.Value = futurePosition;
startKeyframe.KeyTime = KeyTime.FromTimeSpan(TimeSpan.FromSeconds(1));

KeySpline ks = new KeySpline();
ks.ControlPoint1 = new Point(0, 1);
ks.ControlPoint2 = new Point(1, 1);
startKeyframe.KeySpline = ks;
moveAnimation.KeyFrames.Add(startKeyframe);

Storyboard.SetTarget(moveAnimation, currentImage);
Storyboard.SetTargetProperty(moveAnimation, new
➤ PropertyPath("ViewportOrigin"));

moveStoryboard.Children.Add(moveAnimation);
msi.Resources.Add("unique_id", moveStoryboard);

// Play Storyboard
moveStoryboard.Begin();
```

La seule chose à faire auparavant est de réorganiser les images. Pour cela, nous disposons d'une liste d'images qui est renvoyée par notre fonction. Cette liste détermine l'ordre :

```
private List<MultiScaleSubImage> RandomizedListOfImages()
{
    List<MultiScaleSubImage> imageList = new
        ➡ List<MultiScaleSubImage>();
    Random ranNum = new Random();

    // Store List of Images
    foreach (MultiScaleSubImage subImage in msi.SubImages)
    {
        imageList.Add(subImage);
    }

    int numImages = imageList.Count;

    // Randomize Image List
    for (int i = 0; i < numImages; i++)
    {
        MultiScaleSubImage tempImage = imageList[i];
        imageList.RemoveAt(i);

        int ranNumSelect = ranNum.Next(imageList.Count);

        imageList.Insert(ranNumSelect, tempImage);
    }

    return imageList;
}
```

Dans la fonction `ArrangeIntoGrid`, nous récupérons tout au début ces images :

```
List<MultiScaleSubImage> randomList = RandomizedListOfImages();
```

Ensuite, on effectue une boucle sur le nombre de colonnes de notre application (3) et sur le nombre d'images par colonne (nombre d'images/nombre de colonnes - 1) .

Ce sont les seules choses dont vous avez besoin pour créer cette application utilisant *Deepzoom*. Vous pourrez retrouver l'exemple complet dans le code livré avec le livre sur le site de Micro Application.

MouseWheelHelper.cs

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Windows.Browser;

namespace DeepZoomProject
{
    // Courtesy of Pete Blois
    public class MouseWheelEventArgs : EventArgs
    {
        private double delta;
        private bool handled = false;

        public MouseWheelEventArgs(double delta)
        {
            this.delta = delta;
        }

        public double Delta
        {
            get { return this.delta; }
        }

        // Use handled to prevent the default browser behavior!
        public bool Handled
        {
            get { return this.handled; }
            set { this.handled = value; }
        }
    }

    public class MouseWheelHelper
    {
        public event EventHandler<MouseWheelEventArgs> Moved;
        private static Worker worker;
        private bool isMouseOver = false;
    }
}
```

```

public MouseWheelHelper(FrameworkElement element)
{
    if (MouseWheelHelper.worker == null)
        MouseWheelHelper.worker = new Worker();

    MouseWheelHelper.worker.Moved += this.HandleMouseWheel;

    element.MouseEnter += this.HandleMouseEnter;
    element.MouseLeave += this.HandleMouseLeave;
    element.MouseMove += this.HandleMouseMove;
}

private void HandleMouseWheel(object sender,
    ➡ MouseWheelEventArgs args)
{
    if (this.isMouseOver)
        this.Moved(this, args);
}

private void HandleMouseEnter(object sender, EventArgs e)
{
    this.isMouseOver = true;
}

private void HandleMouseLeave(object sender, EventArgs e)
{
    this.isMouseOver = false;
}

private void HandleMouseMove(object sender, EventArgs e)
{
    this.isMouseOver = true;
}

private class Worker
{
    public event EventHandler<MouseWheelEventArgs> Moved;

    public Worker()
    {
        if (HtmlPage.IsEnabled)
        {
            HtmlPage.Window.AttachEvent("DOMMouseScroll",
                ➡ this.HandleMouseWheel);
        }
    }
}

```

```

        HtmlPage.Window.AttachEvent("onmousewheel",
            ➤ this.HandleMouseWheel);
        HtmlPage.Document.AttachEvent("onmousewheel",
            ➤ this.HandleMouseWheel);
    }

}

private void HandleMouseWheel(object sender,
    ➤ HtmlEventArgs args)
{
    double delta = 0;

    ScriptObject eventObj = args.EventObject;

    if (eventObj.GetProperty("wheelDelta") != null)
    {
        delta =
            ➤ ((double)eventObj.GetProperty("wheelDelta"))
            ➤ / 120;

        if (HtmlPage.Window.GetProperty("opera") !=
            ➤ null)
            delta = -delta;
    }
    else if (eventObj.GetProperty("detail") != null)
    {
        delta =
            ➤ -((double)eventObj.GetProperty("detail")) /
            ➤ 3;

        if
            ➤ (HtmlPage.BrowserInformation.UserAgent.IndexOf
            ➤ ("Macintosh") != -1)
            delta = delta * 3;
    }

    if (delta != 0 && this.Moved != null)
    {
        MouseWheelEventArgs wheelArgs = new
            ➤ MouseWheelEventArgs(delta);
        this.Moved(this, wheelArgs);

        if (wheelArgs.Handled)
            args.PreventDefault();
    }
}
}

```

```
    }
}
```

La partie la plus importante est la partie où nous allons attacher les événements :

```
public Worker()
{
    if (HtmlPage.IsEnabled)
    {
        HtmlPage.Window.AttachEvent("DOMMouseScroll",
            ➤ this.HandleMouseWheel);
        HtmlPage.Window.AttachEvent("onmousewheel",
            ➤ this.HandleMouseWheel);
        HtmlPage.Document.AttachEvent("onmousewheel",
            ➤ this.HandleMouseWheel);
    }
}
```

6.4 Deepzoom et Virtual Earth

Virtual Earth est le *Google Map* de Microsoft. Il est dans un sens meilleur que *Google Map* au niveau du nombre de vues qu'il propose par défaut. Son plus gros défaut est d'être payant pour une utilisation poussée.

Si on réfléchit au fonctionnement d'un utilitaire de *map*, on se rend vite compte qu'au final, ce n'est qu'une série d'images mises les unes à côté des autres et les unes au-dessus des autres. On arrive bien à imaginer que cela fonctionne un peu comme *Deepzoom* ou qu'il est très facile d'utiliser *Deepzoom* avec la structure d'images existantes.

Vous pouvez découvrir un projet qui permet d'intégrer *Virtual Earth* dans *Deepzoom*.

Retrouvez ce projet à l'adresse suivante : <http://www.codeplex.com/deepearth>.

6.5 Check-list

Dans ce chapitre sur *Deepzoom*, nous avons étudié :

- ✓ l'élément XAML *MultiScaleImage* ;
- ✓ la manipulation des images dans Deepzoom à l'aide de Deepzoom Composer.



7.1 Silverlight et les langages dynamiques	292
7.2 Introduction au C#	299
7.3 Webographie	311

Annexes

7.1 Silverlight et les langages dynamiques

Silverlight supporte les langages *dynamiques*. Dans ces langages, on retrouve *Python* et *Ruby* que vous connaissez peut-être. On peut facilement imaginer que d'autres langages viendront s'ajouter à cette liste, comme *PHP*.

Le support de ces langages dynamiques permet une meilleure prise en main de *Silverlight*, qui s'appuie alors sur des langages bien plus largement utilisés.

Dans cette annexe, nous verrons comment employer Silverlight avec *Python* et *Ruby*, renommés pour l'occasion *IronPython* et *IronRuby*.

Silverlight et IronPython

Il faut savoir que la *DLR*, outil indispensable pour utiliser les langages dynamiques, est encore en développement. Tous les tests que vous pouvez actuellement effectuer se font sur un composant en développement.

La *DLR* et les langages dynamiques ont été confiés à la communauté (en grande partie). C'est donc une démarche de Microsoft vers l'OpenSource. Cela amène de nombreux avantages mais aussi certains inconvénients.

Ces langages ne sont pas encore bien intégrés à *Visual Studio*. Comme vous allez le voir, nous devons repasser en ligne de commandes pour créer notre projet *IronPython*. Cela devrait être réglé peu de temps après la parution de ce livre dans le courant de 2009.

En attendant, vous devez vous procurer le *SDK* de développement. Ce *SDK* peut être trouvé sur le site de *Silverlight.NET*. Une fois installé, vous pouvez ouvrir votre utilitaire de commande et vous rendre dans le dossier script du fichier téléchargé.

Pour créer un projet, il faut ensuite utiliser cette ligne de commandes :

```
> script/sl [ruby|python|jscript] <application_name>
```

Par exemple :

```
> script/sl python testPythonApp
```

De cette manière, un projet est généré dans un dossier *testPythonApp*. À l'intérieur, vous trouvez quelques fichiers dont un index et deux dossiers. Dans les deux dossiers, vous avez un fichier de style ainsi que deux fichiers composant l'application *Silverlight* : un fichier *XAML* et un fichier *Python*. Vous pouvez les ouvrir avec *Visual Studio* :

```
<UserControl x:Class="System.Windows.Controls.UserControl"
xmlns="http://schemas.microsoft.com/client/2007"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Grid x:Name="layout_root" Background="White">
        <TextBlock x:Name="Message" FontSize="30" />
    </Grid>

</UserControl>
```

Ainsi que le fichier Python :

```
from System.Windows import Application
from System.Windows.Controls import UserControl

class App:
    def __init__(self):
        root = Application.Current.LoadRootVisual(UserControl(), "app.xaml")
        root.Message.Text = "Welcome to Python and Silverlight!"

App()
```

Pour générer ensuite un fichier *xap*, vous avez besoin d'un petit utilitaire appelé *Chiron*. Vous pouvez trouver cet utilitaire dans le package que vous avez téléchargé (*SDK* de la *DLR*). Cet utilitaire s'utilise de la manière suivante :

```
Chiron.exe /directory:MyApp\app /zipdlr:app.xap
```

Nous vous conseillons de déplacer le contenu du fichier *bin* dans le répertoire où vous voulez créer votre fichier *xap*.

Après avoir obtenu ce fichier, vous avez toutes les cartes en main pour créer correctement une application *Silverlight* avec des langages dynamiques.

Une horloge en IronPython et Silverlight

Nous allons créer un simple exemple en *IronPython*. Nous ne reviendrons pas dans cette partie sur la création et la compilation du projet.

Cette application est vraiment simple et composée de deux fichiers : *app.xaml* et *app.py*.

Une horloge est animée, il faut donc déclarer, comme vous l'avez vu au chapitre 2, *Le langage XAML*, une série de triggers et de storyboards :

```
<Canvas.Triggers>
<EventTrigger RoutedEvent="Canvas.Loaded">
    <EventTrigger.Actions>
        <BeginStoryboard>
```

```

<Storyboard>
  <DoubleAnimation x:Name="hourAnimation"
    Storyboard.TargetName="hourHandTransform"
    Storyboard.TargetProperty="Angle"
    From="180" To="540"
    Duration="12:0:0"
    RepeatBehavior="Forever"/>
  <DoubleAnimation x:Name="minuteAnimation"
    Storyboard.TargetName="minuteHandTransform"
    Storyboard.TargetProperty="Angle" From="180"
    To="540" Duration="1:0:0"
    RepeatBehavior="Forever"/>
  <DoubleAnimation x:Name="secondAnimation"
    Storyboard.TargetName="secondHandTransform"
    Storyboard.TargetProperty="Angle" From="180"
    To="540" Duration="0:1:0"
    RepeatBehavior="Forever"/>
  <DoubleAnimation Storyboard.TargetName="parentCanvas"
    Storyboard.TargetProperty="Opacity"
    From="0" To="0.7" Duration="0:0:4"/>
</Storyboard>
</BeginStoryboard>
</EventTrigger.Actions>
</EventTrigger>
</Canvas.Triggers>

```

Une horloge est également caractérisée par une aiguille pour les heures, les minutes et les secondes. Les animations pour ces éléments sont créées dans le storyboard :

```

<!-- Hour hand -->
<Path Data="M -4, 16 1 3 40 3 0 2 -40 z"
  Fill="white">
  <Path.RenderTransform>
    <TransformGroup>
      <RotateTransform x:Name="hourHandTransform"
        Angle="180"/>
      <TranslateTransform X="150.5" Y="145"/>
    </TransformGroup>
  </Path.RenderTransform>
</Path>

<!-- Minute hand -->
<Path Data="M -4, 16 1 3 70 3 0 2 -70 z"
  Fill="white">
  <Path.RenderTransform>
    <TransformGroup>
      <RotateTransform x:Name="minuteHandTransform"

```

```

        Angle="180"/>
        <TranslateTransform X="150.5" Y="145"/>
    </TransformGroup>
</Path.RenderTransform>
</Path>

<!-- Second hand -->
<Path Data="M -1, 16 1 0 70 2 0 0 -70 z" Fill="red">
<Path.RenderTransform>
    <TransformGroup>
        <RotateTransform x:Name="secondHandTransform"
            Angle="180"/>
        <TranslateTransform X="150.5" Y="145"/>
    </TransformGroup>
</Path.RenderTransform>
</Path>

```

Ensuite, il faut donner un décor à notre horloge (la face, une petite ombre, etc.) :

```

<!-- Drop shadow -->
<Path Data="M 157, 5 a 150,150 0 1,0 1,0 z">
<Path.Fill>
    <SolidColorBrush Color="Black" Opacity="0.3"/>
</Path.Fill>
</Path>

<!-- Clock bezel -->
<Path Data="M 150, 0 a 150,150 0 1,0 1,0 z" Fill="black" />
<Path Data="M 150, 1 a 149,149 0 1,0 1,0 z" >
<Path.Fill>
    <LinearGradientBrush>
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="silver" Offset="0.05"/>
                <GradientStop Color="#333333" Offset="0.95"/>
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
</Path.Fill>
</Path>
<Path Data="M 150, 15 a 135,135 0 1,0 1,0 z"
    Fill="black" Opacity="1"/>
<Path Data="M 150, 16 a 134,134 0 1,0 1,0 z"
    Opacity="1">
<Path.Fill>
    <LinearGradientBrush>
        <LinearGradientBrush.GradientStops>

```

```

        <GradientStopCollection>
            <GradientStop Color="#333333" Offset="0.05"/>
            <GradientStop Color="silver" Offset="0.95"/>
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
</Path.Fill>
</Path>

<!-- Clock face -->
<Path Data="M 150, 23 a 127,127 0 1,0 1,0 z"
      Fill="black" Opacity="1"/>

```

Il ne reste plus qu'à ajouter un peu de code Python, qui ira chercher la date actuelle et placer les aiguilles en conséquence :

```

from System.Windows import Application
from System.Windows.Controls import Canvas
from datetime import datetime

class Clock:

    def __init__(self):
        self.scene = Application.Current.LoadRootVisual(Canvas(), "app.xaml")

    def fromAngle(self, time, divisor = 5, offset = 0):
        return ((time / (12.0 * divisor)) * 360) + offset + 180

    def toAngle(self, time):
        return self.fromAngle(time) + 360

    def start(self):
        d = datetime.now()

        self.scene.hourAnimation.From = self.fromAngle(d.hour, 1,
        ➔ d.minute/2)
        self.scene.hourAnimation.To = self.toAngle(d.hour)
        self.scene.minuteAnimation.From = self.fromAngle(d.minute)
        self.scene.minuteAnimation.To = self.toAngle(d.minute)
        self.scene.secondAnimation.From = self.fromAngle(d.second)
        self.scene.secondAnimation.To = self.toAngle(d.second)

Clock().start()

```

L'horloge est terminée et fonctionnelle.

Silverlight et IronRuby

Ruby fait également partie des langages que supporte la *DLR*. Il est très apprécié de la communauté des développeurs pour sa facilité et les nombreux paradigmes qu'il permet de développer (fonctionnel, objet, etc.).

Pour la création d'une application *Ruby* et *Silverlight*, c'est le même principe que pour *Python* (*IronPython*) :

```
> script/sl ruby testRubyApp
```

Seule différence, il y a trois fichiers au niveau du dossier *app*. Le fonctionnement est pourtant le même. Le troisième fichier ajouté (*Silverlight.rb*) est une classe (*SilverlightApplication*) qui permet d'établir la liaison avec le fichier *XAML* :

```
include System::Windows
include System::Windows::Controls
include System::Windows::Media

class SilverlightApplication
  def application
    Application.current
  end

  def self.use_xaml(options = {})
    options = {:type => UserControl, :name => "app"}.merge(options)
    Application.current.load_root_visual(options[:type].new,
    ➤ "#{options[:name]}.xaml")
  end

  def root
    application.root_visual
  end

  def method_missing(m)
    root.send(m)
  end
end

class FrameworkElement
  def method_missing(m)
    find_name(m.to_s.to_clr_string)
  end
end
```

Cette classe sera toujours la même et vous ne devrez probablement jamais rien changer à l'intérieur. Le plus important se trouve dans le fichier *app.rb*.

Une horloge en IronRuby et Silverlight

Il est assez facile d'imaginer que le code *XAML* ne changera pas par rapport à l'application avec *IronPython*. Nous n'allons donc pas tout ressaisir ici.

La seule chose qui change, c'est le code attaché au *XAML*.

Nous devons déterminer l'heure actuelle et placer les aiguilles au bon endroit :

```
require 'Silverlight'

class Clock < SilverlightApplication
  use_xaml :type => Canvas

  def start
    d = Time.now
    root.hour_animation.from = from_angle d.hour, 1, d.minute/2
    root.hour_animation.to = to_angle d.hour
    root.minute_animation.from = from_angle d.minute
    root.minute_animation.to = to_angle d.minute
    root.second_animation.from = from_angle d.second
    root.second_animation.to = to_angle d.second
  end

  def from_angle(time, divisor = 5, offset = 0)
    ((time / (12.0 * divisor)) * 360) + offset + 180
  end

  def to_angle(time)
    from_angle(time) + 360
  end
end

Clock.new.start
```

L'application est terminée.



► Fig. 7.1 :
Notre application

Check-list

Nous avons étudié dans cette annexe :

- Silverlight et les langages dynamiques ;
- IronPython ;
- IronRuby.

7.2 Introduction au C#

Le C# est un langage de programmation à typage fort et orienté objet.

Cela sous-entend :

- *Typage fort*. Chaque variable doit être définie et respecte le type de sa définition. Une variable déclarée comme nombre entier restera un nombre entier tout au long de sa portée et ne pourra utiliser que les méthodes soit appartenant au type *nombre entier* soit utilisant le type *nombre entier*.
- *Orienté objet*. Il est possible de définir de nouveaux types nommés classes. Une classe est une structure de données pouvant contenir des fonctions.

Il existe donc deux sortes de types en C#, les types *primitifs* (nombre entier, nombre réel, chaîne de caractères, etc.) et les types *de classe*.

Une variable instanciée comme étant de *type classe* porte le nom d'*objet*.

Déclaration d'une variable de type primitif

Pour déclarer une variable de type primitif, il suffit d'en écrire son type suivi de son nom et éventuellement sa valeur initiale. Chaque ligne de code suivante est une déclaration de variable valable :



Déclaration de variables de type primitif

```
int nombre ;
string teXte ;
int nombre2 = 4 ;
double nombreRéel = 3.4 ;
string TexTePasVide = " TextePasVide " ;
```

Un type *primitif* commence toujours par une minuscule. Si vous rencontrez un type commençant par une majuscule, c'est qu'il s'agit d'un type *de classe*.

Règles de nommage

Le nom des variables :

- Commence par une lettre ou un caractère de soulignement.
- Indique clairement son contenu.
- Peut contenir invariablement des minuscules, majuscules, des chiffres et des caractères de soulignement. Attention, le *C#* considère comme différentes deux variables du même nom dont seule la case change. (VaRiable1 et Variable1 ne sont pas les mêmes variables.)

La convention de choix du nom de variable la plus courante est le *CamelCase* du nom anglais des chameaux. Tels les bosses d'un chameau, chaque nouveau mot dans le nom d'une variable commence par une majuscule.

Cette convention simplifie grandement la lecture.

Comparez à votre guise ces deux versions du même nom :

- lavariablequivameservirdanslaboucledemonapplicationpourcalculerlatvasurmonsalaire ;
- LaVariableQuiVaMeServirDansLaBoucleDeMonApplicationPourCalculerLaTvaSurMonSalaire.

Il est évident qu'un nom de variable si long est aberrant et ne se retrouvera jamais dans un programme.

Déclaration d'une variable de type de classe

Le mot-clé `new` suivi du nom d'une classe permet de déclarer une nouvelle variable de type de classe.

De nombreuses classes sont fournies par la plateforme *C#*.

Les déclarations de classes suivantes sont correctes :

déclaration de variable de Type de classe

```
Int32 nombre = new Int32();
String s = new String();
String ChaineVide;
```

Attention, dans le cas de la variable de nom `ChaineVide`, seule la déclaration du nom de la variable a été assignée, mais aucune zone mémoire n'a été allouée via le mot-clé `new`.

On dit de la variable `ChaineVide` qu'elle est `null`.

Fonctionnement par référence des types de classe

Les variables de type de classe ne désignent pas uniquement une zone mémoire dans laquelle vous pouvez enregistrer des données.

Il s'agit de pointeur. Un nom de variable pointe une zone mémoire initialisée par le mot-clé `new`.

Il est donc possible de changer la zone mémoire que pointe un nom de variable.

Ainsi dans le code qui suit, `nombre1` va d'abord pointer la zone mémoire définie en ligne 1. Une fois à la ligne 3, `nombre1` pointera la zone mémoire définie en ligne 2, comme `nombre2` :

Zone mémoire et nom de variables

```
1      Int32 nombre1 = new Int32() ;
2      Int32 nombre2 = new Int32() ;
3      nombre1 = nombre2 ;
```

Ce qui est vrai pour les types *de classe* ne l'est pas pour les types *primitifs*. Un type *primitif* représente directement sa zone mémoire. Le même exemple en type *primitif* reviendrait à copier le contenu de la zone mémoire de `nombre2` dans la zone mémoire de `nombre1`.

Revenons à nos types *de classe*. Qu'est-il advenu de la zone mémoire définie en ligne 1 ?

La réponse est simple : elle n'existe plus. Lorsqu'une zone mémoire n'est plus pointée par aucun nom de variable, cette zone mémoire est recyclée par le ramasseur de poubelle de la *plateforme .NET*.

Portée des variables

La portée de vie d'une variable est limitée par les accolades qui entourent le bloc de code dans laquelle elle a été déclarée :



Portée des variables

```
1  {
2      int nombre1 = 3 ;
3      String chaine = new String() ;
4      {
5          String chaine = new String() ;
6          int nombre2 = 4 + nombre1;
7      }
8      {
9          int nombre2 = 5;
10     }
11 }
```

Dans cet exemple :

- La variable `nombre1` est accessible de la ligne 2 à la ligne 10.
- La variable chaîne définie en ligne 3 est accessible :
 - à la ligne 3 ;
 - de la ligne 7 à la ligne 10.

En effet, la variable chaîne déclarée en ligne 5, dans un autre bloc de code, prévaut dans son bloc sur la variable chaîne déclarée en ligne 3.

Cependant, en ligne 5, il est possible d'utiliser la variable `nombre1` définie en ligne 2, car aucune autre variable du nom `nombre1` n'a été définie dans le bloc de la ligne 4 à la ligne 7.

De la même façon, les variables `nombre2` en ligne 6 et `nombre2` en ligne 9 sont deux variables différentes.

Chaque variable est recyclée par le *ramasseur de poubelle* à la fin de son bloc ; la variable `nombre2` de la ligne 6 cesse donc d'exister en ligne 7.

Utilisation des propriétés de classe

Les classes définissent des *objets* pouvant contenir une multitude de données.

Par exemple, la classe `Point` définit un *objet* contenant un double `X` et un double `Y`.

Pour accéder à ces sous-variables, nommées propriétés, il suffit de faire suivre le nom de la variable d'un point et du nom de la propriété :

accès aux propriétés de classe

```
Point p = new Point() ;
p.X = 3.0 ;
p.Y = 4.0 ;
```

Utilisation des méthodes de classe

Une *méthode* est une fonction remplissant un certain usage. Les classes définissent des *objets* pouvant contenir des données mais aussi des *méthodes*.

Exécuter la méthode d'un objet s'effectue en faisant suivre le nom de cet objet par un point, le nom de la méthode et deux parenthèses, une ouvrante et une fermante :

accès aux méthodes de classe

```
Int32 nombre = new Int32() ;
String s = nombre.ToString() ;
```

Dans cet exemple, la méthode `ToString` de la variable `nombre` est appelée. Cette *méthode* convertit la variable en une chaîne de caractères qui sera ensuite assignée à la variable `s`.

Structure d'un programme C# (Partie 1)

Un programme C# débute toujours par l'importation des *librairies* nécessaires à l'exécution de ce programme.

Il y a toujours des *librairies* nécessaires. Par exemple, il serait difficile de se passer de la librairie `System` contenant les types *primitifs*.

Le mot-clé permettant l'importation d'une librairie est `using` ; il est suivi du nom de la librairie voulue.



Exemple de using

```
using Sytem;
using Sytem.Net;
```

Ce qui suit est un *namespace*. Un *namespace* ou *espace de noms* est une sorte d'enclos définissant une famille. Si dans la famille *Boigelot*, le nom de *Simon* signifie *un des auteurs de ce Livre*, dans la famille *Bible*, *Simon* définit *un apôtre*.

Il en va de même pour les espaces de noms.

Ensuite vient une définition de *classe* car en C#, tout est *objet*.

Définir un type de classe

L'avantage réel de la programmation orientée objet n'apparaît que lorsque l'on commence à définir nos propres classes.

Pour rappel, une classe est la définition d'une structure contenant des *propriétés* et des *méthodes*. Cette classe sera instanciée par le mot-clé `new` pour créer des *objets* (zone mémoire) pointés par des noms de variables.

Dans l'autre sens, les noms de variables sont des pointeurs de zone mémoire (objets) respectant une définition contenue dans leur *type de classe*.

Créer une nouvelle *classe* se fait grâce au mot-clé `class`. Toute variable définie dans cette *classe* définit en fait une *propriété* de la classe.

Si nous désirons créer une *classe* représentant des **personnes** et ayant comme *propriété* un **nom** et un **age**, voici ce que la définition de `class` donne :



Définition de la classe Personne (version 1)

```
class Personne
{
    public string Nom ;
    public int Age ;
}
```


Le mot-clé `public` devant chaque *propriété* signifie qu'on peut accéder à cette *propriété* par l'extérieur de la classe. Ainsi il est possible d'écrire :

Déclaration d'un objet de classe `Personne`

```
Personne p = new Personne();
p.Nom = " Simon pas moi mais l'apôtre ";
p.age = 2000;
```

Définir une nouvelle méthode

Une méthode est représentée par son **nom**, son type de **retour** et ses **paramètres** :

- Son nom doit respecter les mêmes règles que ceux des variables.
- Son *type de retour* est le type de résultat de la *méthode*, par exemple pour la méthode `Add` additionnant deux nombres entiers, le type de retour sera un nombre entier.
- Ses *paramètres*, pour la même *méthode* `Add`, les paramètres de la *méthode* seront les deux nombres entiers à additionner.

Cette *méthode* s'écrit :

Méthode `Add`

```
int Add(int nombre1, int nombre2)
{
    return nombre1 + nombre2 ;
}
```

Le mot-clé `return` est suivi du résultat de la *méthode*. En effet, une *méthode* peut être longue et demander de nombreuses lignes de code. Ce mot-clé stipule à la plateforme le résultat à retourner.

Utiliser cette *méthode* s'avère aussi simple mais les *types* de variables des *paramètres* doivent être respectés sous peine d'obtenir une erreur.

Utilisation de la méthode `Add`

```
int ArgentEnPoche = 5 ;
int ArgentEnChaussette = 1 ;
int ArgentTotal = Add(ArgentEnPoche, ArgentEnChaussette) ;
```

Ajouter une méthode à une classe

Pour ajouter une *méthode* à une *classe*, il suffit de définir cette *méthode* à l'intérieur de cette classe.

Par exemple, nous pouvons ajouter la méthode Add à la classe Personne :



Définition de la classe Personne (version 2)

```
class Personne
{
    public string Nom ;
    public int Age ;

    public int Add(int nombre1, int nombre2)
    {
        return nombre1 + nombre2 ;
    }
}
```

Cette *méthode* est elle aussi précédée du mot-clé public pour y accéder depuis l'extérieur de la classe. Il est maintenant possible d'écrire :



Utilisation d'une méthode de classe.

```
Personne p = new Personne();
int somme = p.Add(342,453);
```

Structure d'un programme C# (Partie 2)

Pour en revenir à la structure d'un programme, nous avons vu les using, le namespace et nous étions bloqués sur la définition d'une *classe*.

Dans cette annexe, nous nous contenterons de créer des programmes s'exécutant dans une console dos.

La *classe* de base d'un programme s'exécutant dans une console dos est simplement la class Program.

Son code est :



Code d'une application dos

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace AnnexeConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

La classe Program contient une *méthode* de base, cette *méthode* est la méthode Main, c'est elle qui sera appelée par la plateforme au démarrage du programme.

Elle est précédée du mot-clé static. Ce dernier stipule que cette *méthode* n'est définie qu'une fois à travers toutes les variables.

Il est possible d'accéder aux *méthodes* et propriétés static sans initialiser de variable. Cela se fait en écrivant le nom de la *classe* suivi d'un point et du nom de la *propriété* ou de la *méthode*.

Ainsi, le code de la plateforme exécutant tout programme console est :



code de la plateforme exécutant tout programme console

```
Program.Main(paramètres) ;
```

Exemple d'une application de gestion de données

Nous allons écrire pas à pas une application de gestion de données en mode Console.

Notre première tâche consiste à créer l'application elle-même :



Application de Gestion de données vide

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GestionDeDonnees
{
    class Program
```

```

    {
        static void Main(string[] args)
        {
        }
    }
}

```

Nous devons ensuite définir les données que nous allons gérer. Un bon exemple serait de gérer une *bibliothèque*.

Les différentes *classes* que nous retrouvons dans une application de gestion de *bibliothèques* sont :

- la *bibliothèque* elle-même ;
- des *livres* ;
- des *clients*.

Les différentes actions se déroulant dans une *bibliothèque* sont :

- Un client emprunte un livre pour une certaine durée et paie un certain prix.
- Un client rapporte un livre.

Comme il faut bien commencer, définissons la classe *Livre* :



Livre.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GestionDeDonnees
{
    public class Livre
    {
        public string Titre;
        public string Auteur;
        public DateTime Echeance;
        public int NombreDePage;
        public int Prix;
    }
}

```

Rien que nous n'ayons vu jusqu'à présent. Passons à la classe *Bibliothèque* :

Les listes génériques

Bibliothèque.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GestionDeDonnees
{
    public class Bibliotheque
    {
        public List<Livre> LivreEnMagasin =
            new List<Livre>();
        public List<Client> Clientelle =
            new List<Client>();
    }
}
```

Dans ce cas, la liste attire sans doute votre regard.

En C#, il est possible de déclarer une *propriété* comme étant une liste d'objets. Cette liste n'a pas de taille.

Pour y ajouter un objet, il suffit d'en appeler la *méthode* Add(Objet o).

Ainsi pour ajouter un **Livre** à la *liste* LivreEnMagasin, vous utiliserez le code :

Liste.Add

```
Livre MonLivre = new Livre() ;
Bibliothèque.LivreEnMagasin.Add(MonLivre) ;
```

Pour atteindre un livre dans une liste, on peut utiliser son index, sa position dans la liste :

Utilisation des index dans une liste

```
Livre MonLivre = Bibliothèque.LivreEnMagasin[4]
```

Retirer un livre de la liste se fait grâce à la *méthode* Remove :

Utilisation de List Remove

```
Livre.MonLive = Bibliothèque.LivreEnMagasin[4]
Bibliothèque.LivreEnMagasin.Remove(MonLivre)
```

Ce code donnera le même résultat avec la *méthode* `RemoveAt(Positon)` :



Utilisation de `List.RemoveAt`

```
Bibliothèque.LivreEnMagasin.RemoveAt(4);
```

Il en va de même pour la *classe* `Client` :



Client.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GestionDeDonnees
{
    public class Client
    {
        public string Nom;
        public List<Livre> LivesEnEmprunt =
            new List<Livre>();
    }
}
```

Création des méthode `Emprunte` et `Rend`



Exemple d'utilisation des différentes classes

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GestionDeDonnees
{
    class Program
    {
        private Bibliothèque bibli = new Bibliothèque();

        static void Main(string[] args)
        {
        }
    }
}
```

```

    public void Emprunte(Client client, Livre livre)
    {
        bibli.LivreEnMagasin.Remove(livre);
        client.LivesEnEmprunt.Add(livre);
    }

    public void Rend(Client client, Livre livre)
    {
        client.LivesEnEmprunt.Remove(livre);
        bibli.LivreEnMagasin.Add(livre);
    }
}

```

Conclusion

Le C# est un langage très structurant. Dans les nombreuses librairies offertes par la *plateforme .NET* se trouvent par milliers des définitions de classe, les propriétés et leurs méthodes.

Ces méthodes, empilées couches sur couches, ont ajouté une couche d'abstraction par rapport au langage machine et aux autres langages de première et deuxième génération.

Des méthodes telles que `Mail.Send(Email mail)` ou `monPlayerVideo.PlayVideo(maVideo)` forment la nourriture journalière du développeur C#.

Check-list

Dans cette annexe, nous avons appris les rudiments de la programmation C#, et travers elle, les rudiments de la programmation orientée objet.

Nous espérons que cette annexe aidera celles et ceux d'entre vous qui débutent plein de courage dans ce monde merveilleux de la programmation.

Mais n'oubliez jamais qu'Internet est votre ami. De nombreux tutoriaux sont disponibles sur le sujet pour toutes personnes désireuses d'en apprendre plus.

7.3 Webographie

Ce livre n'aura pas fait de vous un expert. Vous possédez toutes les bases nécessaires pour créer de très bonnes applications *Silverlight*. Mais malheureusement, de Silverlight découle un grand nombre d'autres technologies que nous n'avons pas pu aborder en détail

dans cet ouvrage. C'est pour cela que nous allons vous fournir une série de liens sur différents sujets afin que vous puissiez approfondir vos connaissances.

Visual Studio 2008

Pour respecter l'ordre du livre mais surtout l'ordre logique des choses, il vous faut maîtriser l'outil qui permet la réalisation d'applications *Silverlight*. En effet, nous n'avons pas pris le temps d'expliquer à chaque fois la création de tel ou tel autre fichier, par manque de place.

Vous devez apprendre à maîtriser l'outil.

Sources officielles

Le site de Microsoft qui permet d'obtenir toute l'information sur Visual Studio :

- <http://www.microsoft.com/france/msdn/vstudio/default.mspix>

Une série de vidéos que vous pouvez regarder. Quelques-unes présentent Visual Studio 2010 qui sera le prochain Visual Studio :

- <http://msdn.microsoft.com/fr-fr/vstudio/msdn.5minutes.pour.comprendre.visualstudio.aspx>

Le site de la version gratuite de Visual Studio :

- <http://www.microsoft.com/Express/>

Le portail des développeurs dédié à Visual Studio. Découvrez-y tout ce que voulez savoir sur Visual Studio 2008 :

- <http://msdn.microsoft.com/en-us/vstudio/default.aspx>

Silverlight

Silverlight est le sujet de ce livre. Nous avons abordé de nombreux thèmes mais peut-être reste-t-il des pistes à explorer. Dans tous les cas, il est intéressant de se tenir informé pour un éventuel Silverlight 3 ou la sortie de Silverlight pour application mobile

Sources officielles

Le site officiel de Silverlight. Vous y trouverez des actualités, des démonstrations, etc. À visiter quotidiennement :

- <http://silverlight.net/>

Le site officiel de Silverlight pour les développeurs :

- <http://msdn.microsoft.com/fr-fr/silverlight/default.aspx>

Communautés

Pour vous tenir informer des dernières nouveautés de Silverlight en français :

- <http://www.silverlight-info.fr/>

Le blog de Guillaume, un blogueur à suivre sur le sujet Silverlight et *RIA* :

- <http://blogs.codes-sources.com/guillaume/default.aspx>

Le blog de Christophe Lauer est un bon blog pour se tenir informé sur l'actualité autour de Microsoft, dont celle de Silverlight :

- <http://blogs.msdn.com/clauer/default.aspx>

Le blog de Scott Guthrie est un incontournable pour ceux qui voudraient être au courant de tout à la première minute. Des exemples complets sur les dernières technologies :

- <http://weblogs.asp.net/scottgu/archive/2007/05/07/silverlight.aspx>

Le blog de Jesse Liberty est un blog sur un fan de Silverlight :

- <http://silverlight.net/blogs/jesseliberty/>

On ne saurait les lister tous tellement il y a d'information. Utilisez Google pour davantage de liens.

Le Framework .NET

Voici la dernière et plus grande partie à explorer. Le Framework .NET est un ensemble de technologies qui permettent aussi bien de créer des applications Silverlight, que web ou encore mobile en passant par des applications *Desktop*.

Un très grand nombre de produits gravitent autour de cette technologie qui ne cesse de s'améliorer d'année en année.

Sources officielles

Source officielle de Framework .NET de Microsoft :

- <http://www.microsoft.com/.NET/>

La communauté officielle des développeurs utilisant les technologies .NET :

- <http://www.msdn.com>

Communautés

Très importante communauté autour du .NET. Vous y découvrirez un forum, de l'actualité et un grand nombre d'articles :

- <http://dotnet.developpez.com/>

Codes-Sources est la communauté par défaut où tous les développeurs .NET se rencontrent. Ce site répertorie les derniers postes de blog de la communauté :

- <http://blogs.codes-sources.com/>

Comme son nom l'indique, vous trouverez énormément de code à télécharger sur ce site :

- <http://www.codes-sources.com>

Le blog de Loïc Bar sur toutes les technologies Microsoft :

- <http://blogs.codes-sources.com/loicbar/>

Le blog de Simon Boigelot. Le laboratoire d'un vrai développeur :

- <http://www.simonboigelot.com/>

Voilà qui nous a permis de faire le tour des sites avec lesquels il faut entretenir un contact. Bien entendu, au fur et à mesure de vos visites, vous en découvrirez d'autres.

INDEX

A

ActualHeight	243
ActualWidth	243
ADO.NET	143
Adobe	98
Animations	88
Apache	184
ASMX	135
Asp	
<i>TreeView</i>	195
ASP.NET	97, 182
<i>Membership Provider</i>	197
Assembly	225
Auto	33
AutoGeneratedColumn	253, 256, 264
AutoReverse	82

B

Background	74
BeginInvoke	242
BeginTime	82
Binding	55
Border	33
BorderColor	74
Brush	73
Brushes	73

C

Calendar	247, 249
CallBack	146
Canvas	31

CellEditingTemplate	264
Center	77
CheckBox	42-43
Checked	43 à 45
Clic	39, 54
CodePlex	269
Collapsed	244
ColumnDefinition	28
ComboBox	49
ComboBoxItem	49
CompareValidator	193
Content	40, 45
ContentPresenter	224
ControlTemplate	222-223
Convert	67
ConvertBack	67
CornerRadius	35
CustomValidator	193

D

Data	230
DataBinding	55, 214, 217
DataContext	148, 216-217, 235
DataGrid	253, 255, 264
DataGridCheckBoxColumn	255, 257
DataGridTemplateColumn	255, 257
DataGridTextColumn	255
DataReader	119
DataTemplate	255
DataTemplates	61
DatePicker	249, 263
DateTime	249
Decade	248
Deepzoom	273-274

Default.aspx	184
Default.aspx.cs	184
Dégradé	76
DELETE	116
DependencyProperties	228
Disabled	33
Dispatcher	242
DisplayDate	248-249
DisplayDateEnd	248-249
DisplayDateStart	248-249
DisplayMode	248
Document	24
DOM	210
DoubleAnimationUsingKeyFrame	82
Duration	82
DynamicResource	65

E

Élément	24
EndPoint	76
Événements	38
Expression	
<i>Blend</i>	97
<i>Design</i>	99
<i>Encoder 2</i>	102
<i>Media 2</i>	98
<i>Studio</i>	98
<i>Web</i>	97

F

Flash/Flex	107
Foreground	74

Forever	82
Form	185
FROM	116
FullScreen	243
FullScreenChanged	243

G

Generic.XAML	72
GradientOrigin	77
Gradients	73
Grid	26
GridSplitter	251
GridView	188, 193
GroupName	45

H

Hackers	117
Hidden	33
HorizontalScrollBarVisibility	33
HTML	210

I

Illustrator	98
ImageBrush	78
Images	35
INSERT	116
IronPython	292
IronRuby	297-298
IsChecked	42-43, 45
IsCheked	45
IsFullScreen	242

IsOpen	54
IsSelectedChanged	252
IsThreeState	42, 44-45
ItemPanel	72
ItemSource	62
ItemsSource	262
ItemTemplate	61
IValueConverter	67

K

Key	64
KeyDown	40
KeyUp	40

L

Label	188
LargeChange	51
Layout	26
Line	51
LinearGradientBrush	74
LINQ	113, 126
<i>to Entities</i>	134
<i>to Object</i>	126
<i>to XML</i>	129
ListBox	47, 72
ListBoxItem	47
Live Encoding	103
Login	196
LostFocus	218

M

Margin	35
Maximum	37, 51
MediaElement	237
MediaPlayer	201
MemberShip Provider	197
Microsoft.Windows.Controls.Data	255
Minimum	37, 51
Mode	218
Mounth	248
MouseEnter	40, 90
MouseLeave	40
MouseLeftButtonClic	80
MouseLeftButtonDown	40
MouseLeftButtonUp	40
MouseMouve	90
MouseMove	40
MultiScaleImage	279, 289
MySQL	124
MySqlConnection	125
MySqlDataReader	125

N

Name	38
Namespace	225

O

Objects	
<i>LINQ</i>	126
<i>Timeline</i>	106
ObjectDataSource	193

ODP.NET	120
OneWay	218
Oracle	120
OracleConnection	121
OracleDataReader	122
Orientation	31

P

Password	47
PasswordBox	47
PasswordChar	47
Path	230
Pause	239
Photoshop	98
Plateforme.NET	181
Play	239
Plein écran	242
Popup	53-54
Postback	197
ProgressBar	37
Property	222
PropertyMetadata	229
Provider	127
Python	292

R

RadialGradientBrush	76
RadioButton	45
RadiusX	52
RadiusY	52
RangeValidator	192
RatioValueConverter	70

Rectangle	52
Register	229
RegularExpressionValidator	193
RepeatBehavior	82
Repeater	188, 194
RequiredFieldValidator	192
Ressources	64
RIA	98
RowDefinition	28
RSS	170, 174
Ruby	292

S

ScrollViewer	32
SELECT	116
SelectedDate	249
SelectedDateChanged	249
SelectedItemChanged	49-50
SelectionChanged	252
SGBD	114
Silverlight	97, 201, 207
3	312
<i>pour application mobile</i>	312
Slider	51
SmallChange	51
Soap	162
Source	241
SQL	115-116
SQL Server	114
SQLDataSource	193
StackPanel	30
StartPoint	76
StaticResource	65
Stoke	74

Storyboard	79
Storyboard.TargetName	82
Storyboard.TargetProperty	82
Style	220
System.Windows.Controls	246
System.Windows.Controls.Data	253

T

TabControl	252
TabItem	252
TargetType	222
Template	61, 223, 264
TemplateBinding	223, 230
Text	46-47
TextBlock	36
TextBox	46
TextChanged	46
TextWrapping	37
ToggleButton	43
ToString	60
TreeView	195
TwoWay	218

U

Unchecked	43 à 45
UPDATE	116
URI	35

V

Value	37, 51
ValueChanged	51

ValueConverter	66, 70
VerticalScrollBarVisibility	33
ViewState	197
Virtual Earth	288
Visible	33, 244
Visual Source Safe	109
Visual Studio 2008	109, 183

W

WCF	135
WeatherBugWebServiceSoapClient	162
Web.config	190
Webographie	311
Widget	153
Width	29
Wrap	37
WrapPanel	31

X

XAML	23-24
XML	24
<i>LINQ</i>	129
XMLDataSource	193

Y

Year	248
------------	-----

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

